▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación

23

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación 23

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.

Diseño y maquetación:
BRAVO-LOFISH.

Fotografía:
EQUIPO GALATA.

Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas, Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanera, Licenciada en Informática. APLICACIONES: Sociedad Tamariz, Diplomada en Telecomunicación. OTROS LENGUAJES (COBOL): Eloy Pérez, Licenciado en Informática. Ana Pastor, Licenciada en Informática.

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-160-6
ISBN de la obra: 84-7688-068-7

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M-5-677-1987
Printed in Spain - Impreso en España.

Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.

Agosto, 1987.
P.V.P. Canarias: 335,–.

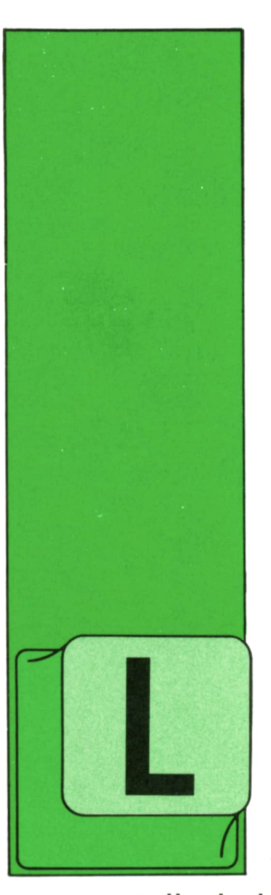
INDICE

BASIC

FUNCIONES (II)



Funciones con argumento alfanumérico



ONGITUD de una cadena: LEN

El formato general es el siguiente:

LEN (N$)

por tanto, acepta un único argumento alfanumérico y da como resultado la longitud de dicho argumento, es decir, el número de caracteres que lo integran.

Veamos algunos ejemplos:

Comando directo	Resultado
PRINT LEN (“EJEMPLO”)	7
LET A$ = “24680”: PRINT LEN (A$)	5
LET A = LEN (“LONGITUD”): PRINT A	8

Valor numérico: VAL

El formato es el siguiente:

VAL (N$)

Acepta un solo argumento alfanumérico que debe cumplir la condición de estar formado exclusivamente por dígitos y opcionalmente un signo (+ o –) y un punto.

Devuelve como resultado la constante numérica correspondiente a ese conjunto de dígitos, es decir, transforma el dato alfanumérico en un dato numérico.

Por ejemplo:

Comando directo	Resultado
PRINT VAL (“1234”)	1234
LET A$ = “23.8”: PRINT VAL (A$)	23.8
LET A = VAL (“–24.5”): PRINT A	–24.5

Existe una función inversa a VAL que se denomina STR$. Esta función la veremos más detenidamente cuando estudiemos las funciones alfanuméricas.

Código ASCII: ASC

El código ASCII es un código existente en todos los ordenadores mediante el cual a cada carácter (letra, número o signo) le corresponde un número. De hecho, todos los caracteres disponibles en nuestro ordenador se almacenan en memoria en forma de número, es decir, representados por su código ASCII correspondiente.

Estos códigos varían desde 0 hasta 255. Del 0 al 31 corresponden a caracteres y teclas de control así como a algunos caracteres gráficos. A partir del código 32 hasta el 126 suelen situarse los caracteres estándar de todos los ordenadores: del 32 al 47 signos diversos de uso común, del 48 al 57 los números naturales (de 0 a 9), del 58 al 64 más signos, del 65 al 90 las letras mayúsculas, del 91 al 96 otro conjunto de signos, del 97 al 122 las letras minúsculas y, finalmente, más signos. A partir del código 127 la mayoría de los ordenadores incluyen diversos caracteres gráficos, que suelen variar mucho de unas máquinas a otras. El SPECTRUM constituye una excepción, ya que, a partir del código 165, incluye todas las palabras BASIC de que dispone. Por otra parte, el COMMODORE incluye caracteres gráficos a partir del código 97, ya que no dispone de minúsculas.

Visto este pequeño resumen del código ASCII, pasamos a analizar la función ASC. Su formato es el siguiente:

ASC (N$)

por tanto, utiliza un único argumento alfanumérico y devuelve como resultado el código ASCII correspondiente al primer carácter del argumento.

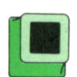
Veamos algunos ejemplos:

Comando directo	Resultado
PRINT ASC ("JAVIER")	74
LET A$ = "E": PRINT ASC (A$)	69
LET A = ASC ("6"): PRINT A	54

En el SPECTRUM esta función se denomina CODE.

Todos los ordenadores disponen de una función inversa de ASC que se denomina CHR$ y que estudiaremos cuando veamos las funciones alfanuméricas.

Por último, puntualizar que las tres funciones analizadas hasta aquí están disponibles en todos los ordenadores, sin excepción.



Funciones alfanuméricas

Conversión a cadena: STR$

Esta función es inversa de VAL y tiene el formato siguiente:

STR$ (N)

por tanto, acepta un único argumento numérico y lo transforma en cadena de caracteres (resultado alfanumérico):

Por ejemplo:

Comando directo	Resultado
PRINT STR$ (42)	"42"
LET A = 32.5: PRINT STR$ (A)	«32.5»
LET A$ = STR$ (−74): PRINT A$	"−74"

Evidentemente las comillas ('') no aparecen, sin embargo, las hemos representado aquí para hacer más patente el carácter alfanumérico del resultado.

Carácter ASCII: CHR$

La función CHR$ es inversa de ASC y tiene el siguiente formato:

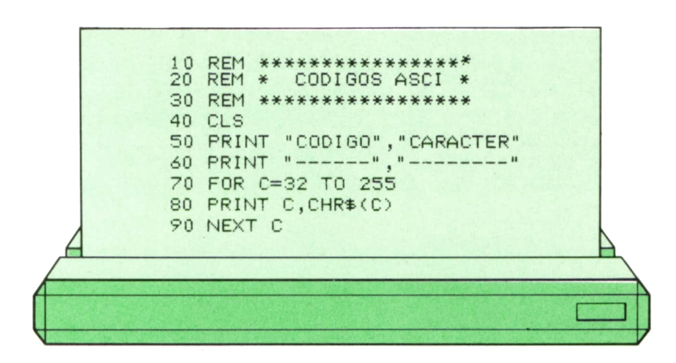
CHR$

Acepta un argumento numérico que debe estar comprendido entre 0 y 255 (intervalo del código ASCII). Devuelve como resultado el carácter correspondiente al código ASCII especificado en el argumento.

Por ejemplo:

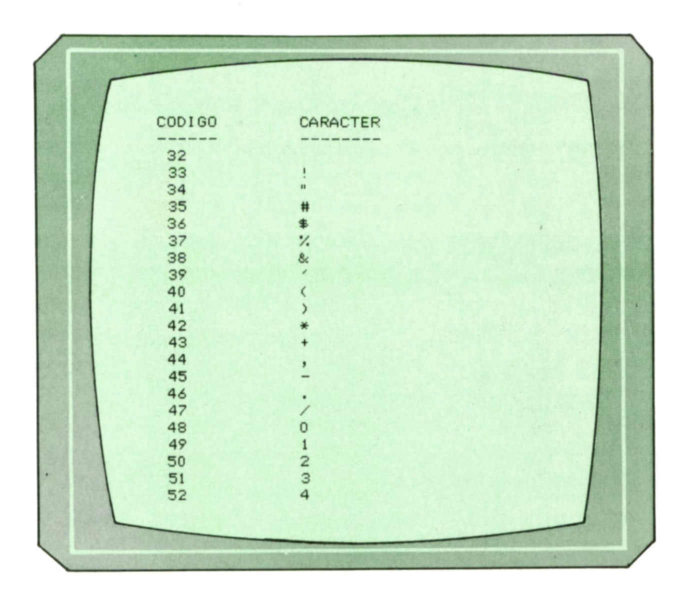
Comando directo	Resultado
PRINT CHR$ (106)	J
LET A = 67: PRINT CHR$ (A)	C
LET A$ = CHR$ (50): PRINT A$	2

El programa 1 nos permite obtener en pantalla la lista de todos los códigos ASCII con sus caracteres correspondientes.



```
10 REM ****************
20 REM *  CODIGOS ASCI *
30 REM ****************
40 CLS
50 PRINT "CODIGO","CARACTER"
60 PRINT "------","--------"
70 FOR C=32 TO 255
80 PRINT C,CHR$(C)
90 NEXT C
```

En la figura 1 podemos ver el aspecto de la pantalla durante la ejecución.



Cadena de caracteres: STRING$

El formato es el siguiente:

STRING$ (N, M$)

por tanto, admite dos argumentos: el primero numérico y el segundo alfanumérico. El resultado es una cadena constituida por tantos caracteres como indique el argumento numérico y tomando como carácter tipo el primer carácter del argumento alfanumérico especificado.

Veamos algunos ejemplos:

Comando directo	Resultado
PRINT STRING$ (4, "=")	====
LET A = 2: LET B$ = "E":	
PRINT STRING$ (A, B$)	EE
LET A$ = STRING$ (5, "JA"):	
PRINT A$	JJJJ

Fragmentación por la izquierda: LEFT$

El formato general es:

LEFT$ (N$, M)

Acepta dos argumentos: el primero alfanumérico y el segundo numérico. Devuelve como resultado una cadena compuesta por los M primeros caracteres de la cadena N$ especificada en el argumento.

Por ejemplo:

Comando directo	Resultado
PRINT LEFT$ ("LIBRERIA", 5)	LIBRE
LET A$ = "ESTAR": LET B = 2: PRINT LEFT$ (A$, B)	ES
LET A$ = LEFT$ ("SILLA", 3): PRINT A$	SIL

Fragmentación por la derecha: RIGHT$

El formato es el siguiente:

RIGHT$ (N$,M)

por tanto, admite dos argumentos, el primero alfanumérico y el segundo numérico. El resultado es una cadena compuesta por los M últimos caracteres de la cadena N$ especificada como argumento.

Veamos algunos ejemplos:

Comando directo	Resultado
PRINT RIGHT$ ("SONRISA",4)	RISA
LET A$="COMILLAS": LET B=6: PRINT RIGHT$(A$,B)	MILLAS
LET A$=RIGHT$("HOLA",3): PRINT A$	OLA

Fragmentación por el centro: MID$

El formato general es el siguiente:

MID$ (N$, M, P)

por tanto, utiliza tres argumentos, el primero alfanumérico y los otros dos numéricos. El resultado es una cadena formada por P caracteres de la cadena N$ especificada como argumento, a partir de la posición M.

Por ejemplo:

Comando directo	Resultado
PRINT MID$ ("GALLINA", 2, 4)	ALLI
LET A$ = "SONIDO": LET B = 3: LET C=2 PRINT MID$ (A$, B, C)	NI
LET A$ = MID$ ("ESCONDER", 3, 5): PRINT A$	CONDE

Fragmentación en el Spectrum

Las tres funciones vistas para fragmentación en cadenas no están disponibles en el SPECTRUM, sin embargo, esto no significa que no sea posible la fragmentación, simplemente es distinta.

El formato es el siguiente:

N$ (M TO P)

es decir, toma de la cadena N$ los caracteres comprendidos entre la posición M y la posición P, siendo P mayor que M. Cuando vayamos a fragmentar por la izquierda no será necesario especificar M (que valdrá 1), y cuando la fragmentación sea por la derecha, no necesitaremos indicar P (que será la posición del último carácter).

Veamos algunos ejemplos:

Comando directo	Resultado
PRINT "VENTANA" (2 TO 5)	ENTA
LET A$ = "SACACORCHOS": PRINT A$ (TO 4)	SACA
LET A$ = "CASCANUECES": LET B = 6: PRINT A$ (B TO)	NUECES

Pulsación de una tecla: INKEY$

Esta es una función alfanumérica especial, ya que no requiere ningún tipo de argumento.

El objetivo de INKEY$ es la lectura instantánea del teclado así como la identificación de la tecla que esté pulsada en ese momento. Si no hay ninguna tecla pulsada la lectura será la cadena vacía ("").

Para que la función INKEY$ resulte útil se suele utilizar de la forma siguiente:

```
10 IF INKEY$ = "" THEN GOTO 10
```

o también la forma:

```
10 LET A$ = INKEY$: IF A$ = "" THEN GOTO 10
```

De este modo producimos un bucle en la línea 10 que obliga a realizar sucesivas lecturas del teclado, hasta que se pulse una tecla cualquiera.

Si añadimos la línea:

```
20 PRINT "Has pulsado la tecla"; A$
```

se imprimirá en pantalla la tecla pulsada.

El COMMODORE dispone de la función GET, que es equivalente a INKEY$. La línea 10 en el COMMODORE se escribiría:

```
10 GET A$: IF A$ = "" THEN GOTO 10
```

Podemos observar que INKEY$ (o GET) e INPUT presentan ciertas analogías: ambas nos permiten introducir un dato por teclado. La diferencia es que INPUT admite datos numéricos o alfanuméricos de cualquier longitud y hay que pulsar INTRO tras teclear dicho dato, mientras que INKEY$ sólo admite datos alfanuméricos de un sólo carácter, y no necesita pulsar INTRO a continuación.

Para finalizar, el programa 2 utiliza varias de las funciones estudiadas en este capítulo.

```
10 REM *************
20 REM *  NOMBRES  *
30 REM *************
40 CLS
50 INPUT "DIME TU NOMBRE";N$
60 CLS
70 PRINT TAB(17);"OPCIONES"
80 PRINT :PRINT :PRINT
90 PRINT TAB(9);"1. INICIAL DEL NOMBRE":PRINT
100 PRINT TAB(9);"2. ULTIMA LETRA DEL NOMBRE":PRINT
110 PRINT TAB(9);"3. NOMBRE EN VERTICAL"
120 PRINT :PRINT :PRINT
130 PRINT TAB(9);"PULSA LA OPCION DESEADA"
140 LET A$=INKEY$:IF A$="" THEN GOTO 140
150 IF ASC(A$)<49 OR ASC(A$)>51 THEN GOTO 140
160 CLS
170 LET A=VAL(A$)
180 ON A GOTO 200,300,400
200 REM * PRIMERA OPCION *
210 PRINT "NOMBRE :",N$
220 LET I$=LEFT$(N$,1)
230 PRINT "INICIAL :",I$
240 GOTO 500
300 REM * SEGUNDA OPCION *
310 PRINT "NOMBRE :",N$
320 LET U$=RIGHT$(N$,1)
330 PRINT "ULTIMA LETRA :",U$
340 GOTO 500
400 REM * TERCERA OPCION *
410 LET L=LEN(N$)
420 FOR I=1 TO L
430 LET C$=MID$(N$,I,1)
440 PRINT C$
450 NEXT I
500 PRINT :PRINT :PRINT
510 PRINT "¿QUIERES VOLVER A EMPEZAR? (S/N)"
520 LET A$=INKEY$:IF A$="" THEN GOTO 520
530 IF A$="S" OR A$="s" THEN GOTO 40
540 IF A$<>"N" AND A$<>"n" THEN GOTO 520
```

En el SPECTRUM habrá que sustituir las siguientes líneas:

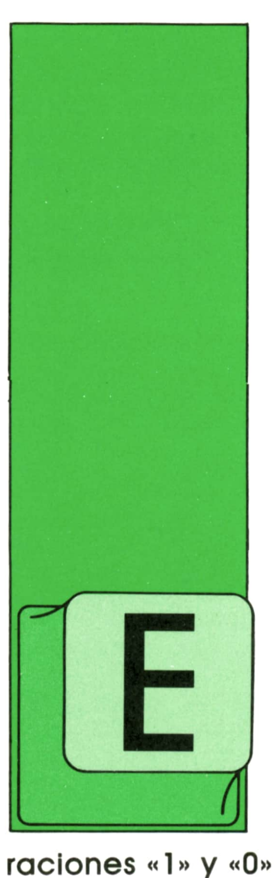
```
150 IF CODE (A$) < 49 OR CODE (A$) >
51 THEN GOTO 140
180 GOTO (A + 1) * 100
220 LET I$ = N$ (1)
320 LET L = LEN N$; LET U$ = N$ (L)
420 LET C$ = N$ (I)
```

MAQUINA 6502

(COMMODORE 64)



Comandos de bifurcación condicionada



ESTOS comandos se utilizan para la toma de decisiones, generalmente detrás de uno de los comandos de comparación. Están basados en las dos posibles configuraciones «1» y «0» de cada uno de los siguientes cuatro flags: N, Z, C y V.

Para que el microprocesador sepa a qué posición debe bifurcar, será necesario un operando que indique la dirección de destino.

Vamos a introducir ahora un tipo especial de direccionamiento llamado «direccionamiento relativo». Es muy sencillo y tan sólo requiere un poco de atención.

Podría tomarse simplemente la dirección de destino, desglosarla en el Low byte (byte bajo) y High byte (byte alto) como ya hemos hecho otras veces, con lo que resultaría un comando de tres bytes, uno para el código del comando y los otros dos para el operando. Sin embargo, la mayor parte de las veces el salto que hemos de hacer no es superior a +/−127 pasos de programa y este número puede ser representado con ocho bits = 1 byte. De esta manera, no se define la dirección a la que se debe saltar, sino la distancia entre la posición actual y la posición después del salto. Esto es lo que se llama direccionamiento relativo.

Muchas veces, sin embargo, será necesario realizar saltos hacia atrás, y tendremos que representar números negativos con la configuración binaria de ocho bits. Si hace un poco de memoria recordará cómo lo hacíamos en la resta binaria cuando hablamos de los números complementarios cuya suma era precisamente 256.

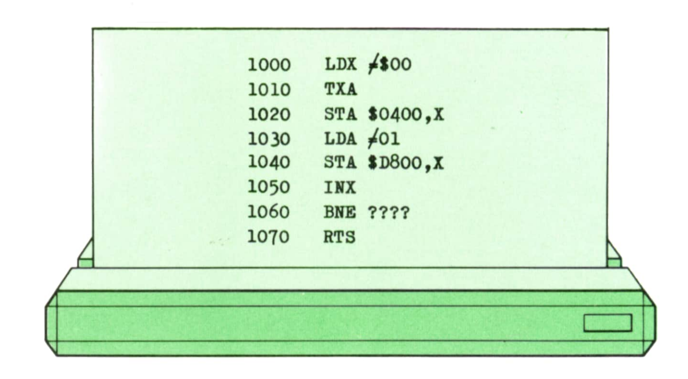
De esta manera si el número «1» se representa %00000001, el número «−1» se representaría como su complementario (los bits a «1» pasan a «0» y viceversa) más «1»: %11111111. El número 2=%00000010 pasaría a -2=%11111110, y así sucesivamente.

Por tanto, el bit 7, o bit más significativo, decide si un número es positivo (0) o negativo (1).

Veamos ahora cómo calcular la distancia en un salto relativo, también llamada OFFSET. Para ello nos vamos a servir de un ejemplo ilustrativo.

Supongamos que queremos hacer una rutina que presente en pantalla el juego de caracteres del COMMODORE 64.

Quedaría una cosa así:



La línea 1000 carga el registro X a cero. La línea 1010 carga el registro ACU con el valor que haya en el registro X. Esto es porque en la línea 1020 hay un direccionamiento indexado que sólo se puede hacer con el ACU. La posición $0400=1024 es la primera correspondiente a la RAM de vídeo. Cuando X se vaya incrementando, también se irá in-

crementando la posición en la que se introduce el dato a partir de la 1024. La línea 1030 carga el ACU a «1», para que el color del carácter siempre sea el mismo, en este caso «amarillo».

La línea 1040 se encarga de ir introduciendo el código del color, a partir de la posición $D800=55296 en adelante según se incremente el valor en X.

La línea 1050 incrementa el valor del registro X en una unidad.

En la línea 1060 queremos que se bifurque siempre que el estado del flag sea diferente de cero. La primera vez que el procesador llega aquí, el valor de X=1, por tanto, bifurcará hasta que llegue al valor X=255, pues después del comando INX, el valor de X=0 y el flag Z=1; entonces ya no bifurca, sino que sigue a la línea 1070 RTS y se acaba la rutina. El problema es qué número habría que poner después de BNE para que el procesador sepa a dónde saltar.

Pues bien, si arrancamos el programa en la dirección $C000, aparecería lo siguiente:

```
$C000   LDX # $00
$C002   TXA
```

Está claro que lo que interesa es saltar al comando TXA cada vez que se pase por la instrucción BNE, es decir, saltar a la posición $C002.

Veamos en qué posición se encuentra el procesador cuando llega a la línea 1060.

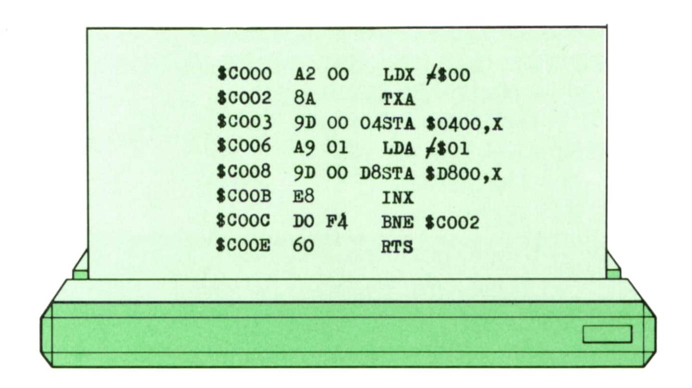
```
$C000  LDX #$00
$C002  TXA
$C003  STA $0400,X
$C006  LDA #$01
$C008  STA $D800,X
$C00B  INX
$C00C  BNE $C002
$C00E  RTS
```

Cuando ha leído la instrucción BNE $C002 se encontrará *DOS* posiciones más adelante de $COOC, es decir, en la posición $COOE.

No se extrañe, pues se trata de un comando de dos bytes y no de tres, ya que la posición $C002, al estar a una distancia menor de 255 bytes (direccionamiento relativo), bastará saber dicha distancia y no la dirección de destino.

Para ello restamos:

```
  $C002
- $C00E
-------
  $FFF4
```



Del resultado nos quedamos con el Low Byte $F4, que es el offset buscado. Si el salto es hacia adelante, se hace una resta normal y corriente y se toma el resultado completo.

Este valor será el que aparezca en el desensamblador del programa. Sin embargo, usted tan sólo debe teclear la dirección de destino $C002, como se indica en el programa anterior, y el ensamblador utilizado para introducir la rutina en C.M. se encargará de calcular el «offset».

Si teclea el programa anterior, y se fija en el desensamblado después, verá que aparece lo siguiente:

Ejecute el programa en lenguaje máquina, y a continuación su equivalente en BASIC, y verá que este último tarda casi siete segundos, mientras que en C.M. apenas lo podrá medir.

¿Qué ventajas proporciona el direccionamiento relativo? Además del ahorro de memoria, pues se trata de un comando de dos bytes, y la consiguiente rapidez de ejecución, hay una cuestión muy importante: al microprocesador se le indica únicamente la distancia que hay desde donde actualmente se encuentra hasta la posición a la que debe saltar. Así un programa en el que se modifiquen, añadan o borren instrucciones *fuera* del intervalo de salto, no necesita una modificación del programa dentro del intervalo, pues la distancia entre los extremos permanece inalterada.

Lo único que se debe tener en cuenta a la hora de utilizar el direccionamiento relativo es que tan sólo se pueden realizar saltos de 129 bytes adelante y 126 bytes hacia atrás.

Tampoco debe preocuparle el cálculo de la distancia entre las dos posiciones de memoria, es trabajo del programa ensamblador que está utilizando.

Diremos que existen bifurcaciones según el estado de los flags:

1) Z - Zero flag
2) N - Negative
3) C - Carry flag
4) V - Flag V

1) Cuando se quiere bifurcar con el flag Z activado se utiliza el comando BEQ, del inglés «Branch on Equal». Es decir, cuando utilizamos un comando de comparación, si el resultado es igual a cero, y a continuación introducimos un BEQ, el microprocesador saltará a la dirección especificada. Si el resultado de la comparación es diferente de cero, el microprocesador seguirá a la siguiente dirección, no bifurcará.

Si hay que bifurcar cuando el flag Z esté desactivado, se utilizará el comando BNE «Branch on Not Equal».

2) La bifurcación con el flag N activado es BMI «Branch on Minus», y con el flag N desactivado BPL «Branch on Plus».

3) Para bifurcar con el Carry flag activado se utiliza el comando BCS «Branch on Carry Set», y para hacerlo con el Carry flag desactivado, BCC «Branch on Carry Clear».

4) El flag overflow V, también puede utilizarse como base para las bifurcaciones condicionadas. Para la bifurcación cuando está activado se usa el comando BVS «Branch on Overflow Set» y cuando esté desactivado BVC «Branch on Overflow Clear».

La siguiente tabla ofrece un resumen de los comandos expuestos:

Comando	Flag base	Código de comando
BEQ	Z	$F0
BNE	Z	$D0
BCS	C	$B0
BCC	C	$90
BMI	N	$30
BPL	N	$10
BVS	V	$70
BVC	V	$50



Comandos para la bifurcación incondicional

Cuando el procesador se encuentra una instrucción de este tipo salta *siempre* a la dirección especificada en el comando sin que se tenga que cumplir ningún tipo de condición. Es similar al GOTO del BASIC.

Además del direccionamiento absoluto existe el direccionamiento indirecto, utilizando dos posiciones contiguas en la memoria que forman un apuntador hacia la dirección de destino. El comando es JMP.

Veamos un ejemplo de cada modo:

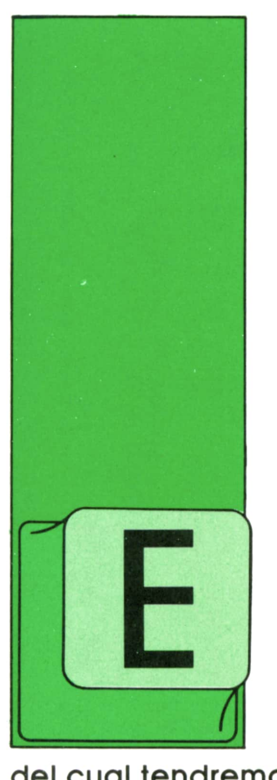
JMP $C000 salta a la dirección $C000
JMP ($0301)

En este caso el procesador toma el dato en la posición $0301 en forma de LB o byte bajo. A continuación toma el dato de la posición siguiente $0302 en forma de HB o byte alto, y forma así un apuntador a $HBLB. Es a esta posición a la que se produce el salto.

El código para JMP en direccionamiento absoluto es $4C, y el código para el direccionamiento indirecto es $6C.

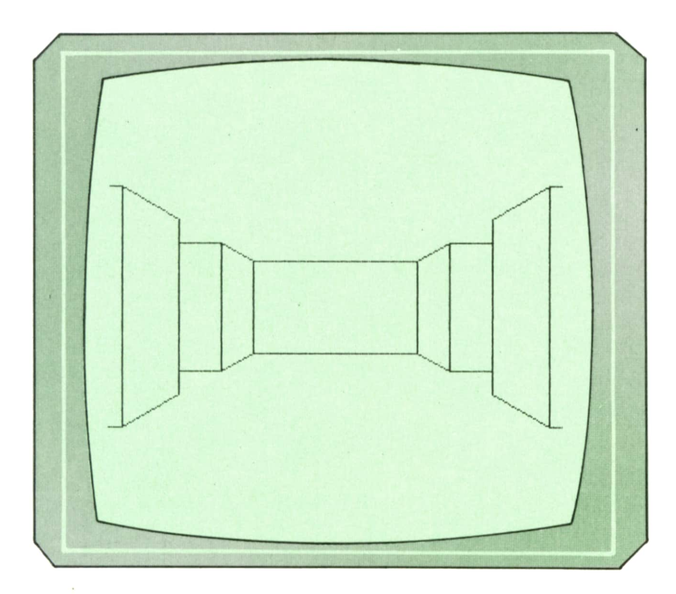
PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS



Laberinto para SPECTRUM

ESTE pequeño, pero emocionante juego, servirá para poner a prueba nuestro sentido de la orientación. Al ejecutar el juego apareceremos dentro de un laberinto, del cual tendremos que salir en el menor tiempo posible. Dicho laberinto cambiará de un juego a otro.

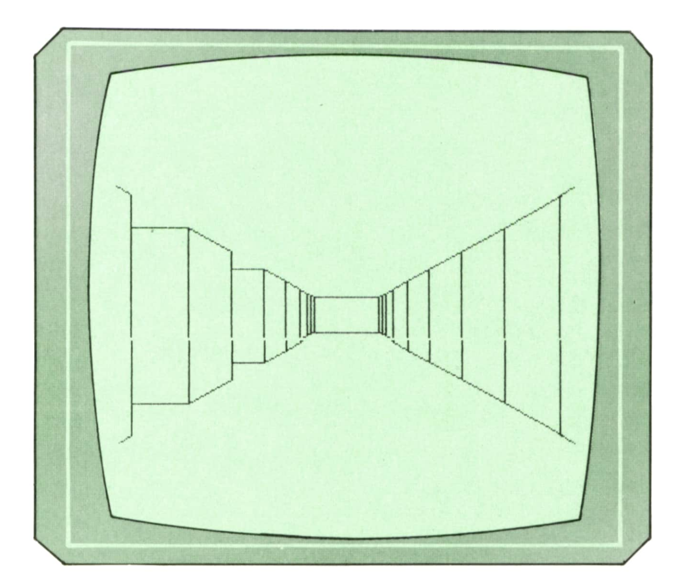




Estamos dentro del laberinto y enfrente tenemos una pared. A la derecha tenemos dos callejones y a la izquierda otros dos.

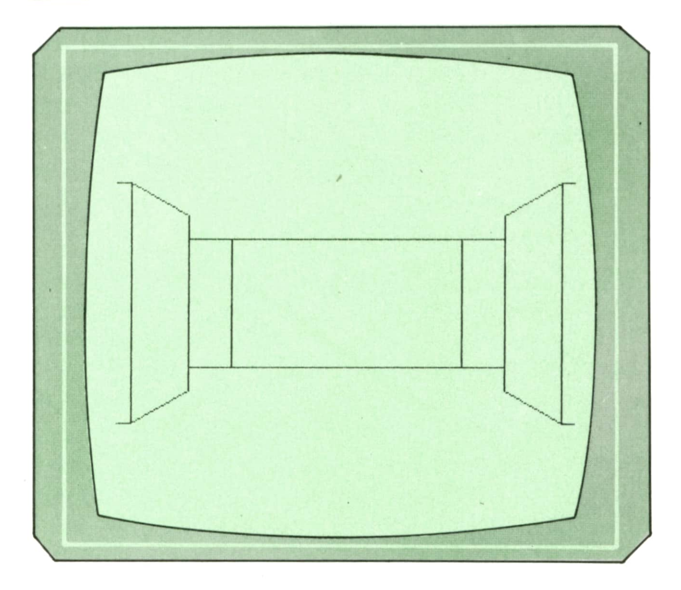
Para movernos por el laberinto tenemos que utilizar las siguientes teclas:

SPACE — Para avanzar.
O — Para girar a la izquierda.
P — Para girar a la derecha.
R — Para darnos la vuelta y mirar lo que hay a nuestras espaldas.





Tenemos dos pasillos a la izquierda.

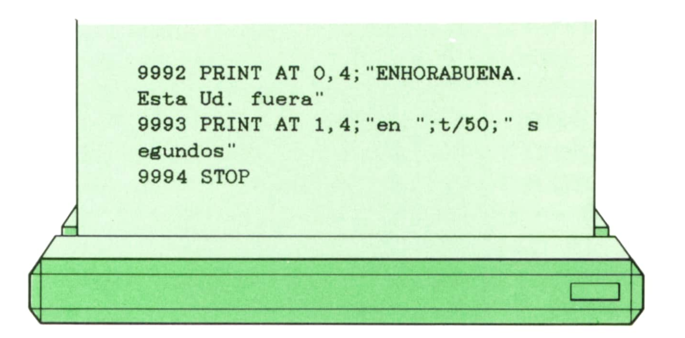




Nos encontramos en una encrucijada.

```
   1 REM
   2 REM LABERINTO
   3 REM
 100 DIM v(10,10): DIM h(10,10)
 110 GO SUB 9000
 120 LET x=1: LET y=1: LET dx=1:
 LET dy=0
 125 LET ti=PEEK 23672+256*PEEK
23673+4096*PEEK 23674
 130 LET L=9: LET Lx=x+(dx=-1):
LET Ly=y+(dy=-1)
 132 LET L=L-1: LET Lx=Lx+dx: LE
T Ly=Ly+dy
 134 IF dx<>0 AND v(Lx,Ly)=0 THE
N  GO TO 132
 136 IF dy<>0 AND h(Lx,Ly)=0 THE
N  GO TO 132
 140 CLS : GO SUB 6000+L
 145 LET Lx=Lx-(dx=-1): LET Ly=L
y-(dy=-1)
 150 FOR i=L TO 8
 155 LET Lx=Lx-dx: LET Ly=Ly-dy:
 GO SUB 1000: NEXT i
 156 IF dx=-1 AND y=9 AND x<5 TH
EN  PRINT AT 10,14;"SALIDA"
 158 LET a$=INKEY$: IF a$="" THE
N  GO TO 158
 160 IF a$<>" " THEN  GO TO 190
 170 IF (dx=1 AND v(x+1,y)=0) OR
 (dx=-1 AND v(x,y)=0) OR (dy=1 A
ND h(x,y+1)=0) OR (dy=-1 AND h(x
,y)=0) THEN  LET x=x+dx: LET y=y
+dy
 190 IF x=1 AND y=9 THEN  GO TO
9990
 200 IF a$="r" THEN  LET dx=-dx:
 LET dy=-dy: GO TO 130
 210 IF a$="o" THEN  GO TO 300
 215 IF a$<>"p" THEN  GO TO 130
 220 IF ABS dx=1 THEN  LET dy=-d
x: LET dx=0: GO TO 130
 230 LET dx=dy: LET dy=0: GO TO
130
 300 IF ABS dx=1 THEN  LET dy=dx
: LET dx=0: GO TO 130
 310 LET dx=-dy: LET dy=0: GO TO
 130
1000 IF dx<>1 THEN  GO TO 1100
1010 IF h(Lx,Ly+1)=0 THEN  GO SU
B 7500+i: GO TO 1050
1020 GO SUB 7000+i
1050 IF h(Lx,Ly)=0 THEN  GO TO 8
500+i
1060 GO TO 8000+i
1100 IF dx<>-1 THEN  GO TO 1200
1110 IF h(Lx,Ly)=0 THEN  GO SUB
7500+i: GO TO 1150
1120 GO SUB 7000+i
1150 IF h(Lx,Ly)=0 THEN  GO TO 8
500+i
1160 GO TO 8000+i
1200 IF dy<>-1 THEN  GO TO 1300
1210 IF v(Lx+1,Ly)=0 THEN  GO SU
B 7500+i: GO TO 1250
1220 GO SUB 7000+i
1250 IF v(Lx,Ly)=0 THEN  GO TO 8
500+i
1260 GO TO 8000+i
1300 IF v(Lx,Ly)=0 THEN  GO SUB
7500+i: GO TO 1350
1310 GO SUB 7000+i
1350 IF v(Lx+1,Ly)=0 THEN  GO TO
 8500+i
1360 GO TO 8000+i
6000 PLOT 110,76: DRAW 0,24: DRA
W 36,0: DRAW 0,-24: DRAW -36,0:
RETURN
6001 PLOT 108,75: DRAW 0,26: DRA
W 40,0: DRAW 0,-26: DRAW -40,0:
RETURN
6002 PLOT 106,74: DRAW 0,28: DRA
W 44,0: DRAW 0,-28: DRAW -44,0:
RETURN
6003 PLOT 102,71: DRAW 0,34: DRA
W 52,0: DRAW 0,-34: DRAW -52,0:
RETURN
6004 PLOT 94,65: DRAW 0,46: DRAW
 68,0: DRAW 0,-46: DRAW -68,0: R
ETURN
6005 PLOT 82,57: DRAW 0,62: DRAW
 92,0: DRAW 0,-62: DRAW -92,0: R
ETURN
6006 PLOT 64,45: DRAW 0,86: DRAW
 128,0: DRAW 0,-86: DRAW -128,0:
 RETURN
6007 PLOT 40,29: DRAW 0,118: DRA
W 176,0: DRAW 0,-118: DRAW -176,
0: RETURN
6008 PLOT 8,7: DRAW 0,162: DRAW
240,0: DRAW 0,-162: DRAW -240,0:
 RETURN
7000 PLOT 108,75: DRAW 2,1: DRAW
 0,24: DRAW -2,1: RETURN
7001 PLOT 106,74: DRAW 2,1: DRAW
 0,26: DRAW -2,1: RETURN
7002 PLOT 102,71: DRAW 4,3: DRAW
 0,28: DRAW -4,3: RETURN
7003 PLOT 94,65: DRAW 8,6: DRAW
0,34: DRAW -8,6: RETURN
7004 PLOT 82,57: DRAW 12,8: DRAW
 0,46: DRAW -12,8: RETURN
7005 PLOT 64,45: DRAW 18,12: DRA
W 0,62: DRAW -18,12: RETURN
7006 PLOT 40,29: DRAW 24,16: DRA
W 0,86: DRAW -24,16: RETURN
7007 PLOT 8,7: DRAW 32,22: DRAW
0,118: DRAW -32,22: RETURN
7008 PLOT 0,1: DRAW 8,6: DRAW 0,
162: DRAW -8,6: RETURN
7500 PLOT 108,76: DRAW 2,0: DRAW
 0,24: DRAW -2,0: RETURN
7501 PLOT 106,75: DRAW 2,0: DRAW
 0,26: DRAW -2,0: RETURN
7502 PLOT 102,74: DRAW 4,0: DRAW
 0,28: DRAW -4,0: RETURN
7503 PLOT 94,71: DRAW 8,0: DRAW
0,34: DRAW -8,0: RETURN
7504 PLOT 82,65: DRAW 12,0: DRAW
 0,46: DRAW -12,0: RETURN
7505 PLOT 64,57: DRAW 18,0: DRAW
 0,62: DRAW -18,0: RETURN
7506 PLOT 40,45: DRAW 24,0: DRAW
 0,86: DRAW -24,0: RETURN
7507 PLOT 8,29: DRAW 32,0: DRAW
0,118: DRAW -32,0: RETURN
```

```
7508 PLOT 0,7: DRAW 8,0: DRAW 0,
162: DRAW -8,0: RETURN
8000 PLOT 148,75: DRAW -2,1: DRA
W 0,24: DRAW 2,1: RETURN
8001 PLOT 150,74: DRAW -2,1: DRA
W 0,26: DRAW 2,1: RETURN
8002 PLOT 154,71: DRAW -4,3: DRA
W 0,28: DRAW 4,3: RETURN
8003 PLOT 162,65: DRAW -8,6: DRA
W 0,34: DRAW 8,6: RETURN
8004 PLOT 174,57: DRAW -12,8: DR
AW 0,46: DRAW 12,8: RETURN
8005 PLOT 192,45: DRAW -18,12: D
RAW 0,62: DRAW 18,12: RETURN
8006 PLOT 216,29: DRAW -24,16: D
RAW 0,86: DRAW 24,16: RETURN
8007 PLOT 248,7: DRAW -32,22: DR
AW 0,118: DRAW 32,22: RETURN
8008 PLOT 255,1: DRAW -8,6: DRAW
 0,162: DRAW 8,6: RETURN
8500 PLOT 148,76: DRAW -2,0: DRA
W 0,24: DRAW 2,0: RETURN
8501 PLOT 150,75: DRAW -2,0: DRA
W 0,26: DRAW 2,0: RETURN
8502 PLOT 154,74: DRAW -4,0: DRA
W 0,28: DRAW 4,0: RETURN
8503 PLOT 162,71: DRAW -8,0: DRA
W 0,34: DRAW 8,0: RETURN
8504 PLOT 174,65: DRAW -12,0: DR
AW 0,46: DRAW 12,0: RETURN
8505 PLOT 192,57: DRAW -18,0: DR
AW 0,62: DRAW 18,0: RETURN
8506 PLOT 216,45: DRAW -24,0: DR
AW 0,86: DRAW 24,0: RETURN
8507 PLOT 248,29: DRAW -32,0: DR
AW 0,118: DRAW 32,0: RETURN
8508 PLOT 255,7: DRAW -7,0: DRAW
 0,162: DRAW 7,0: RETURN
9000 FOR i=1 TO 9: LET h(i,1)=1:
 LET h(i,10)=1: FOR j=2 TO 9: RE
AD a: LET h(i,j)=a: NEXT j: NEXT
 i
9010 FOR i=1 TO 9: LET v(1,i)=1:
 LET v(10,i)=1: FOR j=2 TO 9: RE
AD a: LET v(j,i)=a: NEXT j: NEXT
 i
9020 RETURN
9100 DATA 1,1,1,0,0,0,0,1
9101 DATA 0,0,1,0,0,1,0,1
9102 DATA 1,1,0,0,0,0,1,1
9103 DATA 0,0,1,1,0,0,1,0
9104 DATA 1,1,2,0,0,1,1,0
9105 DATA 1,0,0,0,0,0,0,1
9106 DATA 1,1,1,0,1,0,0,1
9107 DATA 1,1,1,0,0,1,1,0
9108 DATA 1,0,0,0,0,0,0,0
9200 DATA 0,0,0,0,0,0,0,0
9201 DATA 1,1,1,1,0,1,0,0
9202 DATA 0,0,0,0,0,0,0,1
9203 DATA 0,1,0,0,1,0,0,0
9204 DATA 1,1,0,1,1,1,1,1
9205 DATA 1,1,1,1,1,1,0,1
9206 DATA 0,0,0,0,0,0,1,0
9207 DATA 1,1,0,0,1,1,0,0
9208 DATA 0,0,0,1,0,0,0,1
9990 LET te=PEEK 23672+256*PEEK
23673+4096*PEEK 23674
9991 LET t=te-ti
```



```
9992 PRINT AT 0,4;"ENHORABUENA.
Esta Ud. fuera"
9993 PRINT AT 1,4;"en ";t/50;" s
egundos"
9994 STOP
```

Editor de caracteres para MSX

Con la ayuda de este programa podremos definir nuestro propio juego de caracteres en el MSX. Estos caracteres sólo podrán ser utilizados en el modo 1 de pantalla (SCREEN 1).

Una vez que el programa esté en memoria y tecleemos la palabra RUN, nos aparecerá en pantalla una parrilla en la que tendremos que definir nuestros caracteres. También nos aparecerá un pequeño menú de ayuda que nos dirá qué funciones nos permite realizar el programa.

Dentro de la parrilla veremos a un cursor parpadeando. Este cursor lo tendremos que ir moviendo por la parrilla para ir definiendo el carácter que queremos realizar. Para moverlo tenemos que utilizar las teclas del cursor (o el joystick si se tiene). Una vez que estemos sobre un punto que queremos iluminar (o apagar) tenemos que pulsar la barra espaciadora. Si dicho punto estaba apagado, se iluminará, y si estaba iluminado se apagará.

El resto de las funciones que podemos utilizar están disponibles pulsando las teclas de función. Son las siguientes:

F1 — INVIERTE EL CARACTER. Todos los puntos que estén iluminados se apagan y todos los que estén apagados se iluminan.

F2 — ROTA A LA IZQUIERDA. El carácter entero sufre un desplazamiento de un punto a la izquierda. La parte del carácter que desaparezca por la izquierda, reaparecerá por la derecha.

F3 — ROTA A LA DERECHA. Igual que el anterior, pero en este caso el desplazamiento es hacia la derecha. Lo que desaparezca por la derecha reaparecerá por la izquierda.

F4 — ESPEJO HORIZONTAL. El carácter se nos aparece como si lo estuviésemos viendo reflejado en un espejo. Lo que antes estaba a la derecha ahora está en la izquierda y lo que estaba a la izquierda ahora está a la derecha.

F5 — ESPEJO VERTICAL. El efecto resultante después de pulsar esta opción es como si el carácter se hubiese dado la vuelta. Todo lo que antes estaba arriba ahora está abajo, y lo que estaba abajo ahora está arriba.

F6 — LOAD. Sirve para leer un fichero que previamente hubiésemos grabado en cassette con todos los caracteres definidos. Para poder leer el fichero, éste tiene que haber sido grabado con la función que hay en la tecla F7.

F7 — SAVE SET INTERNO. Nos sirve para grabar todos los caracteres que hemos definido de manera que más tarde podamos volver a leerlos con la tecla F6 para su modificación o terminación.

F8 — SAVE EN LINEAS DE DATA. Pulsando esta opción, el ordenador nos grabará los caracteres que hemos definido almacenados en líneas DATA. De esta manera, si queremos utilizar dichos caracteres en nuestros programas, sólo necesitamos realizar un MERGE para tener dichas líneas en nuestro programa.

F9 — EDITA UN CARACTER. Sirve para editar un carácter de los que el ordenador tiene almacenados. Esta opción nos puede servir para modificar los caracteres que tiene el MSX.

F10 — BORRA PARRILLA. El efecto que obtenemos al pulsar esta opción es el del borrado de la parrilla.

Una vez que hemos terminado de definir los caracteres que queríamos crear, para unirlos a nuestro programa, hay que realizar los siguientes pasos:

1. Con nuestro programa en memoria, realizar un MERGE de las líneas DATA.

2. Incluir las siguientes líneas en nuestro programa (los números de línea son arbitrarios y tendrás que poner los números de línea que mejor vayan con tu programa):

```
100 RESTORE N.º de línea donde empieza
la primera línea DATA.
110 LET N = BASE (7)
120 FOR I = n1 TO n2
130       FOR J = 0 TO 7
140             READ A
150             VPOKE N + 8 * I + J, A
160       NEXT J
170 NEXT I
```

donde ni es el primer carácter que queremos definir y n2 el último.

Lo que hace este grupo de líneas es leer los números de las líneas DATA y almacenarlos en la memoria del vídeo del MSX.

```
1000 REM ***********************
1010 REM *                     *
1020 REM *    E D I T O R      *
1030 REM *                     *
1040 REM *        D E          *
1050 REM *                     *
1060 REM * C A R A C T E R E S *
1070 REM *                     *
1080 REM *      P A R A        *
1090 REM *                     *
1100 REM *       M S X         *
1110 REM *                     *
1120 REM ***********************
1130 REM
1140 REM *************************
1150 REM *************************
1160 REM * (c) ED. SIGLO CULTURAL *
1170 REM *************************
1180 REM * (c) 1987               *
1190 REM *************************
1200 REM *************************
1210 REM
1220 SCREEN 1
1230 LOCATE 0,0,0
1240 COLOR 11,0,0
1250 CLEAR 300
```

```
1260 KEY OFF
1270 PRINT "    EDITOR DE CARACTERES"
1280 PRINT "  ========================"
1290 LOCATE 1,12
1300 PRINT "ESPERE UN MOMENTO POR FAVOR"
1310 DIM C(255,7)
1320 DI=BASE(7)
1330 FOR G=0 TO 255
1340    FOR I=0 TO 7
1350       C(G,I)=VPEEK(G*8+DI+I)
1360    NEXT I
1370 NEXT G
1380 REM
1390 REM ***************************
1400 REM * DEFINICION DE VARIABLES *
1410 REM ***************************
1420 REM
1430 MU=BA+173*8
1440 F=0
1450 FOR I=DI+171*8 TO DI+171*8+15
1460    READ A
1470    VPOKEI,A
1480 NEXT I
1490 DATA 0,126,66,66,66,66,126,0,0,126,126,126,126,126,126,0
1500 CLS
1510 LOCATE 15,3
1520 PRINT"CARACTER :"
1530 PU=65
1540 CX=0
1550 CY=0
1560 GOSUB 2450
1570 GOSUB 2150
1580 REM
1590 REM ******************
1600 REM * MENU PRINCIPAL *
1610 REM ******************
1620 REM
1630 LOCATE 0,9
1640 PRINT"Opciones:"
1650 PRINT
1660 PRINT"F1-Invierte.","F6-Load.","F2-Rota iz.","F7-Save set.","F3-Rota der.",
"F8-Save data.","F4-Espejo hor.F9-Edita.","F5-Espejo ver.F10-Borra.",STRING$(29,
"-")
1670 GOSUB 1950
1680 REM
1690 REM ********************
1700 REM * FORMA DE CONTROL *
1710 REM *                  *
1720 REM * 0=TECLADO        *
1730 REM * 1=JOYSTICK       *
1740 REM ********************
1750 REM
1760 FC=0
1770 ON STRIG GOSUB 2360
1780 STRIG(FC) ON
1790 ON INTERVAL=25 GOSUB 2250
1800 ON KEY GOSUB 2920,2700,2810,3160,3030,3330,3560,3770,2550,4030
1810 FOR I=1 TO 10
1820    KEY(I) ON
1830 NEXT I
1840 M=STICK(FC)
1850 IF M<>0 THEN STRIG(FC) OFF:INTERVAL STOP:F=1:GOSUB 2250 ELSE 1840
1860 FOR CC=1 TO 40
1870 NEXT CC
1880 CY=(CY+(M=1)-(M=5) AND 7)
1890 CX=(CX-(M=3)+(M=7) AND 7)
1900 F=1
1910 GOSUB 2250
```

```
1920 INTERVAL ON
1930 STRIG(FC) ON
1940 GOTO 1840
1950 REM
1960 REM *****************
1970 REM * PONE PARRILLA *
1980 REM *****************
1990 REM
2000 INTERVAL OFF
2010 STRIG(FC) OFF
2020 LOCATE 0,0,0
2030 FOR G=0 TO 7
2040    T=128
2050    FOR N=0 TO 7
2060       IF (C(PU,G) AND T)=T THEN PRINTCHR$(172); ELSE PRINTCHR$(171);
2070       T=T/2
2080    NEXT N
2090    PRINT
2100 NEXT G
2110 INTERVAL ON
2120 STRIG(FC) ON
2130 BEEP
2140 RETURN
2150 REM
2160 REM *********************
2170 REM * ASIGNA PARRILLA AL *
2180 REM * CARACTER 173.      *
2190 REM *********************
2200 REM
2210 FOR G=0 TO 7
2220    VPOKE MU+G,C(PU,G)
2230 NEXT G
2240 RETURN
2250 REM
2260 REM ************
2270 REM * INTERVAL *
2280 REM ************
2290 REM
2300 LOCATE CX,CY,0
2310 T$=" "
2320 IF F THEN IF (C(PU,CY) AND 2^(7-CX))<>0 THEN T$=CHR$(172) ELSE T$=CHR$(171)
2330 PRINT T$
2340 F=-(F=0)
2350 RETURN
2360 REM
2370 REM ***********
2380 REM * TRIGGER *
2390 REM ***********
2400 REM
2410 VPOKE MU+CY,(VPEEK(MU+CY) XOR 2^(7-CX))
2420 C(PU,CY)=VPEEK(MU+CY)
2430 RETURN
2440 REM
2450 REM *******************
2460 REM * PONE EL CARACTER *
2470 REM * ASIGNADO         *
2480 REM *******************
2490 REM
2500 INTERVAL OFF
2510 LOCATE 13,5
2520 PRINT PU;"  ";TAB(18);"-";CHR$(PU);"-";CHR$(173)
2530 INTERVAL ON
2540 RETURN
2550 REM
2560 REM *********************
2570 REM * EDITA EL CARACTER *
2580 REM * ELEGIDO.          *
2590 REM *********************
```

```
2600 REM
2610 GOSUB 4340
2620 LOCATE 0,17
2630 INPUT"Caracter a editar";A$
2640 PU=ASC(A$)
2650 GOSUB 2170
2660 GOSUB 2450
2670 GOSUB 1950
2680 GOSUB 4260
2690 RETURN
2700 REM
2710 REM **********************
2720 REM * ROTACION IZQUIERDA *
2730 REM **********************
2740 REM
2750 FOR I=0 TO 7
2760    C(PU,I)=(C(PU,I)*2 AND 254)+(C(PU,I) AND 128)/128
2770    VPOKE MU+I,C(PU,I)
2780 NEXT I
2790 GOSUB 1950
2800 RETURN
2810 REM
2820 REM ********************
2830 REM * ROTACION DERECHA *
2840 REM ********************
2850 REM
2860 FOR I=0 TO 7
2870    C(PU,I)=(C(PU,I)/2 AND 127)+128*(C(PU,I) AND 1)
2880    VPOKE MU+I,C(PU,I)
2890 NEXT I
2900 GOSUB 1950
2910 RETURN
2920 REM
2930 REM **************************
2940 REM * INVERSION DEL CARACTER *
2950 REM **************************
2960 REM
2970 FOR G=0 TO 7
2980    C(PU,G)=(C(PU,G) XOR 255)
2990    VPOKE MU+G,C(PU,G)
3000 NEXT G
3010 GOSUB 1950
3020 RETURN
3030 REM
3040 REM ******************
3050 REM * ESPEJO VERTICAL *
3060 REM ******************
3070 REM
3080 FOR G=0 TO 3
3090    T=C(PU,7-G)
3100    C(PU,7-G)=C(PU,G)
3110    C(PU,G)=T
3120 NEXT G
3130 GOSUB 1950
3140 GOSUB 2150
3150 RETURN
3160 REM
3170 REM *********************
3180 REM * ESPEJO HORIZONTAL *
3190 REM *********************
3200 REM
3210 FOR G=0 TO 7
3220 B$=BIN$(C(PU,G))
3230 B$=STRING$(8-LEN(B$),"0")+B$
3240    A$=""
3250    FOR I=1 TO 8
3260       A$=MID$(B$,I,1)+A$
3270    NEXT I
```

```
3280    C(PU,G)=VAL("&B"+A$)
3290    VPOKE MU+G,C(PU,G)
3300 NEXT G
3310 GOSUB 1950
3320 RETURN
3330 REM
3340 REM ******************
3350 REM * LOAD CARACTERES *
3360 REM ******************
3370 REM
3380 GOSUB 4340
3390 LOCATE 0,17
3400 INPUT "A partir del caracter ";C$
3410 INPUT "Nombre";N$
3420 T=ASC(C$)
3430 PRINT"Ponga en marcha el cassette.."
3440 OPEN "cas:"+N$ FOR INPUT AS #1
3450 IF EOF(1) THEN 3510
3460 FOR I=0 TO 7
3470    INPUT#1,C(T,I)
3480 NEXT I
3490 T=T+1
3500 GOTO 3450
3510 CLOSE#1
3520 GOSUB 4240
3530 GOSUB 1950
3540 GOSUB 2150
3550 RETURN
3560 REM
3570 REM ********
3580 REM * SAVE *
3590 REM ********
3600 REM
3610 GOSUB 4340
3620 LOCATE 0,17
3630 INPUT"A partir del caracter";C$
3640 T=ASC(C$)
3650 INPUT"Cuantos caracteres";N
3660 INPUT "Nombre";N$
3670 GOSUB 4140
3680 OPEN "cas:"+N$ FOR OUTPUT AS #1
3690 FOR G=T TO T+N-1
3700    FOR I=0 TO 7
3710       PRINT#1,C(G,I)
3720    NEXT I
3730 NEXT G
3740 CLOSE#1
3750 GOSUB 4240
3760 RETURN
3770 REM
3780 REM ******************
3790 REM * SAVE COMO DATAS *
3800 REM ******************
3810 REM
3820 GOSUB 4340
3830 LOCATE 0,17
3840 INPUT"A partir del caracter";N$
3850 T=ASC(N$)
3860 INPUT "Cuantos caracteres";N
3870 INPUT "Nombre";N$
3880 INPUT "Linea de comienzo";LI
3890 GOSUB 4140
3900 OPEN "cas:"+N$ FOR OUTPUT AS #1
3910 FOR G=T TO T+N-1
3920    A$=STR$(LI)+" DATA "
3930    FOR I=0 TO 6
3940       A$=A$+"&H"+HEX$(C(G,I))+","
3950    NEXT I
```

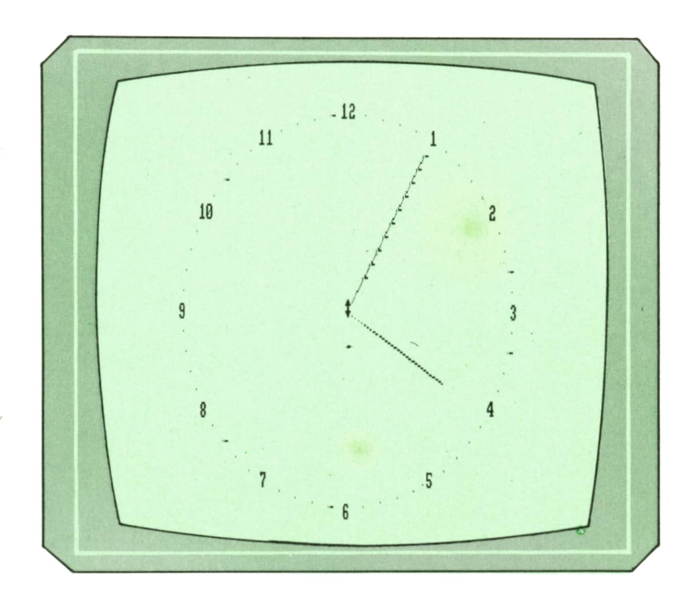
```
3960    A$=A$+"&H"+HEX$(C(G,7))
3970    PRINT#1,A$
3980    LI=LI+10
3990 NEXT G
4000 CLOSE#1
4010 GOSUB 4240
4020 RETURN
4030 REM
4040 REM ***********************
4050 REM * BORRADO DE CARACTER *
4060 REM ***********************
4070 REM
4080 FOR G=0 TO 7
4090    C(PU,G)=0
4100    VPOKE MU+G,0
4110 NEXT G
4120 GOSUB 1950
4130 RETURN
4140 REM
4150 REM ***********************
4160 REM * MENSAJE DE CASSETTE *
4170 REM ***********************
4180 REM
4190 LOCATE 0,22
4200 PRINT"Ponga en marcha el cassette        y pulse una tecla";
4210 I$=INKEY$
4220 IF I$="" THEN 4210
4230 RETURN
4240 REM
4250 REM *******************
4260 REM * BORRA LA VENTANA *
4270 REM * INFERIOR         *
4280 REM *******************
4290 REM
4300 LOCATE 29,23
4310 PRINT STRING$(203,127)
4320 INTERVAL ON
4330 RETURN
4340 REM
4350 REM ****************
4360 REM * PREPARA INPUT *
4370 REM ****************
4380 REM
4390 POKE &HF3F8,PEEK(&HF3FA)
4400 POKE &HF3F9,PEEK(&HF3FB)
4410 INTERVAL OFF
4420 RETURN
```



Reloj educativo para AMSTRAD

Este pequeño programa pensado para los más pequeños de la casa nos servirá para poder enseñarles cómo es un reloj y cómo leer la hora.

Para mover las manecillas del reloj sólo tenemos que pulsar la tecla del cursor. La tecla del cursor a la derecha hace que aumente la hora, y cursor a la izquierda que disminuya.

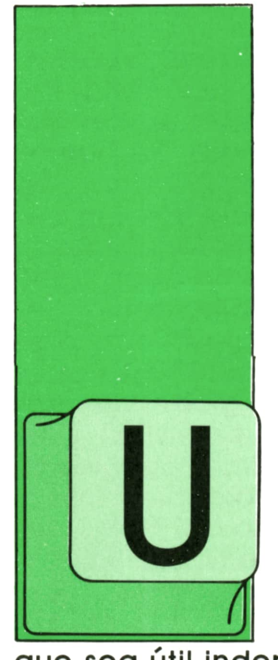



```
10 REM *****************************
20 REM ***     RELOJ EDUCATIVO     ***
30 REM *** Un programa realizado ***
40 REM ***          Por            ***
50 REM *** Carlos A. Maria Morin ***
60 REM ***                         ***
70 REM ***      (C) Ediciones      ***
80 REM ***  Siglo Cultural 1987    ***
90 REM *****************************
100 REM
110 MODE 2
120 DEG
130 m=6
140 n=2.5
150 SYMBOL 255,0,0,0,60,126,255,126,60
160 ORIGIN 320,200
170 FOR pu=0 TO 359 STEP 6
180 PLOT 175*SIN(pu),175*COS(pu),5
190 NEXT
200 TAG
210 ORIGIN 0,0
220 MOVE 312,385
230 PRINT"12";
240 MOVE 406,360
250 PRINT "1";
260 MOVE 469,295
270 PRINT "2";
280 MOVE 492,207
290 PRINT"3";
300 MOVE 469,120
310 PRINT "4";
320 MOVE 404,57
330 PRINT "5";
340 MOVE 316,29
350 PRINT"6";
360 MOVE 228,57
370 PRINT "7";
380 MOVE 164,118
390 PRINT "8";
400 MOVE 141,207
410 PRINT"9";
420 MOVE 162,294
430 PRINT "10";
440 MOVE 225,360
```

```
450 PRINT "11";
460 ORIGIN 320,200
470 MOVE -3,10
480 PRINT CHR$(255);
490 IF INKEY(8)=0 THEN a=a+6:g=g+6:m=6:n=2.5
500 IF INKEY(1)=0 THEN a=a+30:g=g+30:m=30:n=2.5
510 IF g=30 THEN c=c+2.5:g=0
520 MOVE 0,0
530 DRAW 120*SIN(c),120*COS(c),1
540 MOVE 0,0
550 DRAW 160*SIN(a),160*COS(a),1
560 MOVE 0,0
570 DRAW 160*SIN(a-m),160*COS(a-m),0
580 MOVE 0,0
590 DRAW 120*SIN(c),120*COS(c),1
600 MOVE 0,0
610 DRAW 120*SIN(c-n),120*COS(c-n),0
620 MOVE 0,0
630 DRAW 160*SIN(a),160*COS(a),1
640 GOTO 470
```

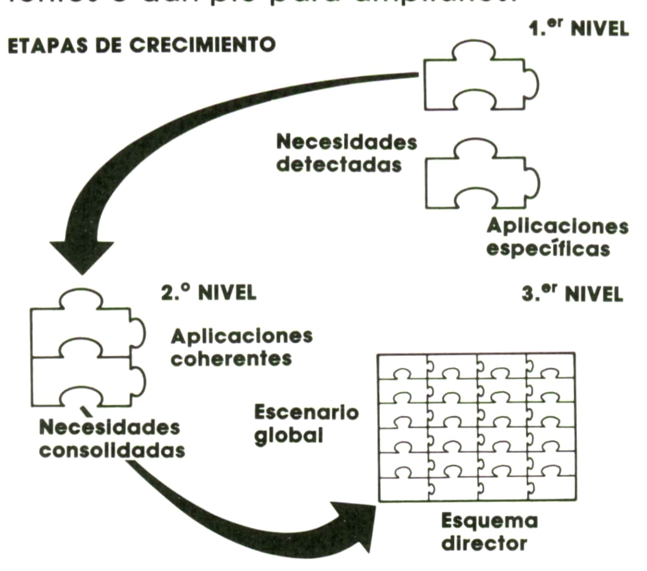
TECNICAS DE ANALISIS

DIFERENTES ETAPAS DEL DESARROLLO DE PROYECTOS INFORMATICOS



NA de las ventajas del método Merise es, precisamente, la facilidad que aporta para descomponer las tareas a abordar en el proyecto informático, de tal modo que sea útil independientemente del nivel de mecanización de la Empresa (del nivel de informatización conseguido) y del tamaño del proyecto que se desee abordar. Se pueden definir tres niveles de crecimiento en la informatización de una Organización.

a) **Primer nivel.** Este primer nivel de arranque es el de las Empresas que están en fase de introducción de la Informática en su Organización. Los proyectos se ocupan de automatizar localmente o perfeccionar la automatización de algún proceso administrativo (facturación, nóminas, gestión de ventas, etc.) sin entrar en las reglas de funcionamiento. Estos proyectos utilizan las capacidades residuales de los medios informáticos existentes o dan pie para ampliarlos.



En este primer nivel es útil introducir un **plan informático.** Este plan consiste, básicamente, en planificar los proyectos para el siguiente año según las necesidades manifestadas por los usuarios; prevé el aumento necesario de capacidad de las máquinas disponibles o las nuevas compras a realizar.

Cada proyecto no supone ninguna innovación importante sobre las reglas de gestión establecidas o respecto de las informaciones que se manejan. La elaboración del dossier de especificaciones no tiene en cuenta, por tanto, ningún parámetro desconocido.

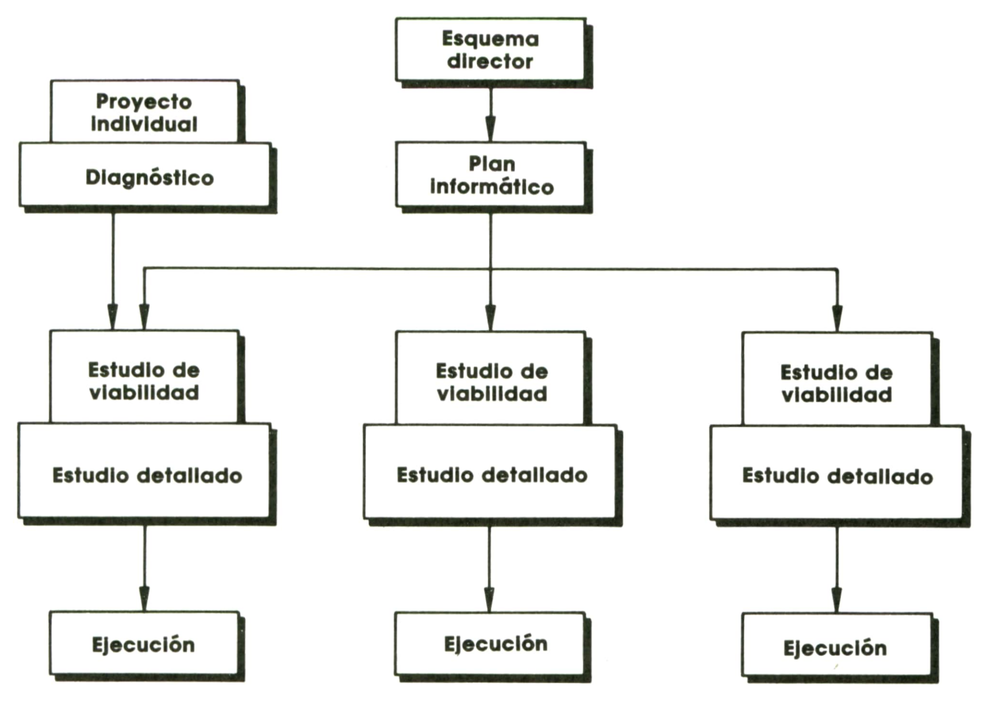
b) **Segundo nivel.** En esta etapa tanto los informáticos como los usuarios se preocupan de la coherencia y comunicación de las diferentes aplicaciones: se examinan a fondo ciertos procesos de gestión. Por ello, la elaboración de los proyectos informáticos pasa a ser más delicada, la duración de los estudios se alarga y los costes se elevan. Es necesario en este caso adoptar un punto de vista más «industrial» y considerar el proyecto a realizar como una etapa más del desarrollo de la empresa. Se impone en este caso realizar un cuidadoso estudio económico. Se desarrolla un plan en dos etapas: elaboración de un estudio de viabilidad y preparación posterior de un estudio detallado.

El **estudio de viabilidad** cubre el conjunto del área a la que se refiere el proyecto (por ejemplo, gestión de personal, gestión comercial, etc.). En este estudio se incluye un plan global (lo que se llama un «plan-masa») que contiene todas las soluciones futuras en el área involucrada, con descomposición del proyecto general en sub-proyectos y con el es-


Evolución de la «informatización» de la Empresa.

tablecimiento de un presupuesto provisional. Si el equipo directivo aprueba este estudio de viabilidad se van lanzando subproyectos dentro del plan informático establecido. Cada uno de estos subproyectos da lugar a un **estudio detallado** que conduce al establecimiento del dossier de especificaciones de detalle correspondiente. Es importante en este estudio incluir los análisis necesarios en cuanto a costes y elaboración de los calendarios correspondientes, según las limitaciones del «plan-mosa» definido y aprobado.

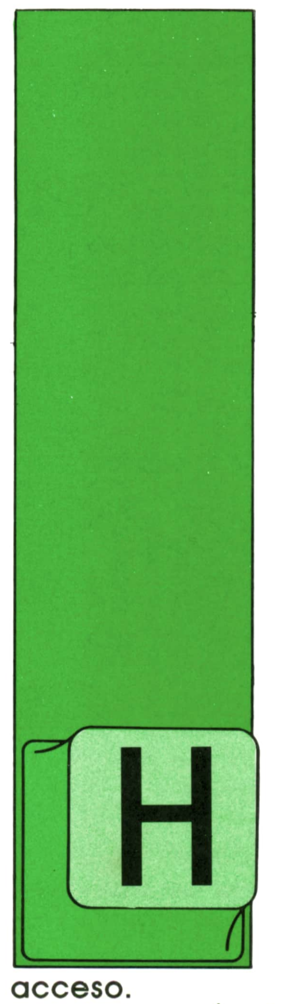
Naturalmente, en cualquier momento, independientemente del nivel de mecanización alcanzado pueden surgir necesidades específicas a resolver mediante un pequeño proyecto adicional. Se suele establecer en este caso un **«informe-diagnóstico»** para precisar la naturaleza y límites del problema a abordar: su coste, el calendario de su evolución y las implicaciones que pueda tener con el resto de los proyectos definidos. Se puede hacer un test simple ante una situación de este tipo: ¿el problema está bien definido?, ¿están claramente enunciados los objetivos funcionales y aceptados por todas las personas involucradas?, ¿el proyecto se integra en algún otro proyecto de mayor envergadura?, ¿se puede hacer una evaluación global de ventajas aportadas y costes previstos? Si la contestación a estas cuatro preguntas es positiva, se puede pasar directamente a hacer el estudio detallado del proyecto. En caso contrario, merece la pena abordar previamente la realización de un estudio de viabilidad.



Desarrollo de los estudios informáticos.

c) **Tercer nivel.** Se llega a este nivel cuando ya es una realidad la «informatización» de la Empresa, las aplicaciones existentes funcionan y se relacionan en alguna medida incipiente, se quieren modificar varios de los sistemas en funcionamiento, y el conjunto puede dejar de ser controlable en cualquier momento. Los responsables se sienten incapaces de tener en cuenta las diversas circunstancias de cada subconjunto y se impone el análisis global de la Organización. En estas circunstancias suele ser conveniente concebir y analizar un escenario global previsible de evolución de los sistemas de información de la Empresa y, sobre él, llegar a definir un **esquema-director.** En este tipo de estudio se definen, con un horizonte de cinco a diez años, las grandes orientaciones de las organizaciones, los ciclos de evolución de las aplicaciones y el desarrollo general de los bancos de información de la Compañía. Este esquema director es una magnífica herramienta de gestión del patrimonio informático de la Empresa.

TECNICAS DE PROGRAMACIÓN





Instrucciones de transferencia incondicional

HEMOS visto cómo es posible dar nombre a una instrucción para hacerla accesible desde otra parte del programa. Ahora vamos a ver cómo se lleva a cabo dicho acceso.

La instrucción de transferencia incondicional indica al ordenador que la ejecución secuencial de las instrucciones del programa debe detenerse en este momento, saltar a la instrucción cuyo nombre se le indica y continuar secuencialmente a partir de ese punto, hasta que se encuentre con una nueva instrucción de transferencia o con una instrucción de bucle, que son las únicas que modifican la marcha secuencial de los programas.

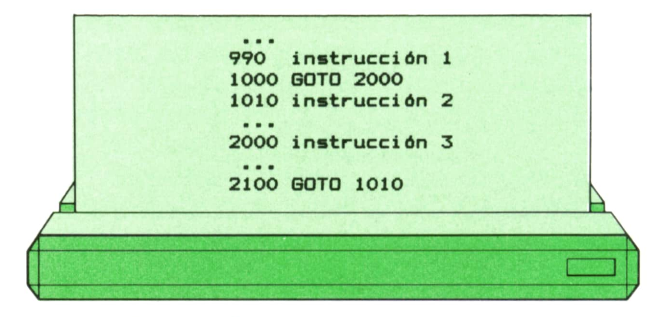
En BASIC y PASCAL la instrucción de transferencia incondicional tiene la forma:

GOTO etiqueta

donde «etiqueta» es el nombre de la instrucción a donde deseamos saltar, mientras que «GOTO» es una contracción de las palabras inglesas «GO TO», cuyo equivalente castellano es «IR A». Naturalmente, en PASCAL habrá que añadir el punto y coma final en los casos en que sea necesario.

Más adelante veremos qué forma adopta esta instrucción en el lenguaje APL.

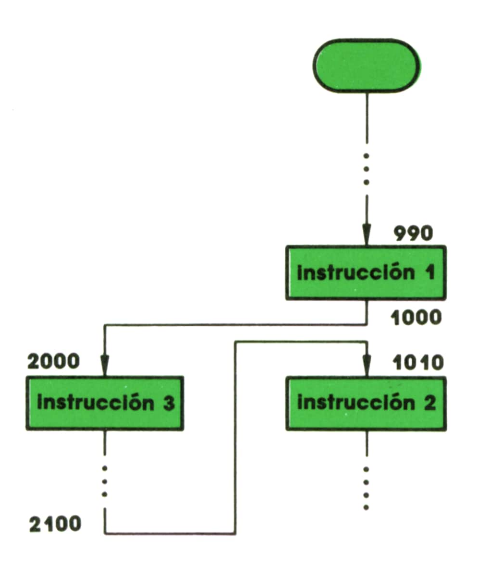
Veamos un ejemplo de un programa en BASIC que utiliza la instrucción GOTO:



```
...
990  instrucción 1
1000 GOTO 2000
1010 instrucción 2
...
2000 instrucción 3
...
2100 GOTO 1010
```

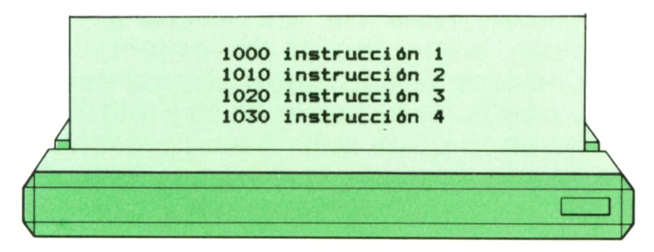
cuya ejecución tiene lugar de la siguiente manera: en cierto momento, durante la ejecución del programa, podrá ejecutarse la instrucción de etiqueta 990 (instrucción 1). A continuación le toca ejecutarse a la instrucción de etiqueta 1000, pero lo que ésta hace es transferir la ejecución a la instrucción de etiqueta 2000 (instrucción 3). Después se ejecutarán secuencialmente algunas instrucciones más, hasta que, al llegar a la de etiqueta 2100, encontramos una nueva instrucción de transferencia incondicional, que nos envía a la instrucción de etiqueta 1010 (instrucción 2). A partir de este momento la ejecución continúa sucesivamente . Se observará, por tanto, que el orden en que se han ejecutado las tres instrucciones (instrucción 1, 3 y 2) es diferente de aquél en que han sido definidas (instrucción 1, 2 y 3).

Veamos el organigrama de este programa, en el que hemos añadido la etiqueta de cada instrucción para facilitar su identificación:



Puede comprobarse que las instrucciones de transferencia no aparecen explícitamente en los organigramas de los programas, sino que se convierten simplemente en líneas que unen dos puntos del diagrama arbitrariamente alejados entre sí.

Es conveniente utilizar con cuidado las instrucciones de transferencia incondicional, porque la libertad que proporcionan para alcanzar las instrucciones localizadas en un punto y otro del programa puede llevar a la fragmentación de éste y hacerlo casi ininteligible, incluso para su autor. Veamos un ejemplo. Sea el programa

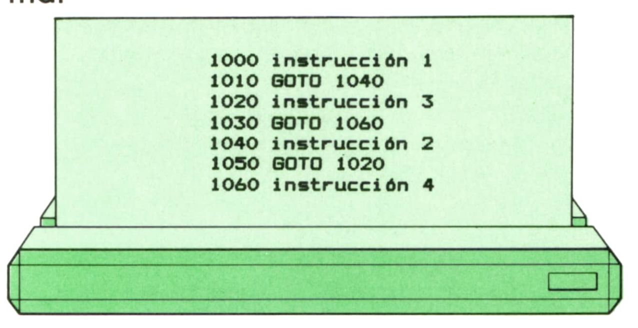


```
1000 instrucción 1
1010 instrucción 2
1020 instrucción 3
1030 instrucción 4
```

cuyo organigrama es



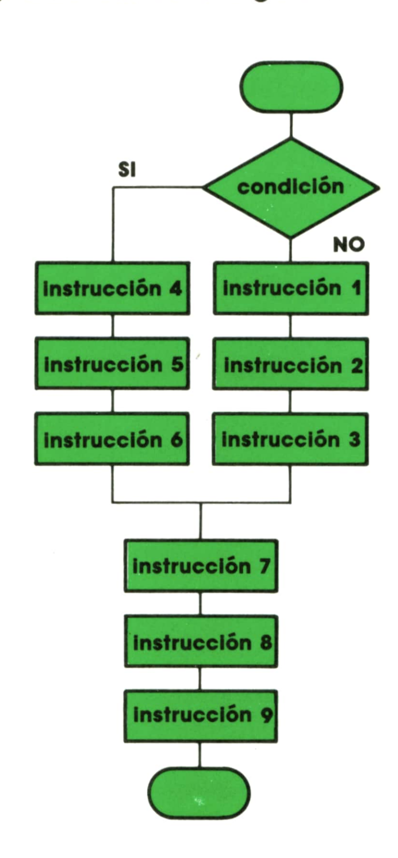
Observemos ahora el siguiente programa:



```
1000 instrucción 1
1010 GOTO 1040
1020 instrucción 3
1030 GOTO 1060
1040 instrucción 2
1050 GOTO 1020
1060 instrucción 4
```

No es difícil darse cuenta de que su organigrama es el mismo que el del programa anterior. Pero también es evidente que su estructura es mucho menos clara que la de aquél. Puede comprenderse fácilmente hasta qué punto un programa puede hacerse completamente ilegible si se abusa de las instrucciones de transferencia incondicional.

Uno de los casos en que suele aprovecharse con eficacia la instrucción de tansferencia incondicional en el lenguaje BASIC, se presenta cuando un programa, cuya marcha se ha bifurcado por cualquier motivo en dos caminos paralelos entre sí, debe reunirlos de nuevo en uno solo. Consideremos, por ejemplo, un organigrama como el siguiente:



En este programa, si se cumple la condición, se ejecutarán sucesivamente las instrucciones 4, 5, 6, 7, 8 y 9. Si no se cumple, se ejecutarán las instrucciones 1, 2, 3, 7, 8 y 9.

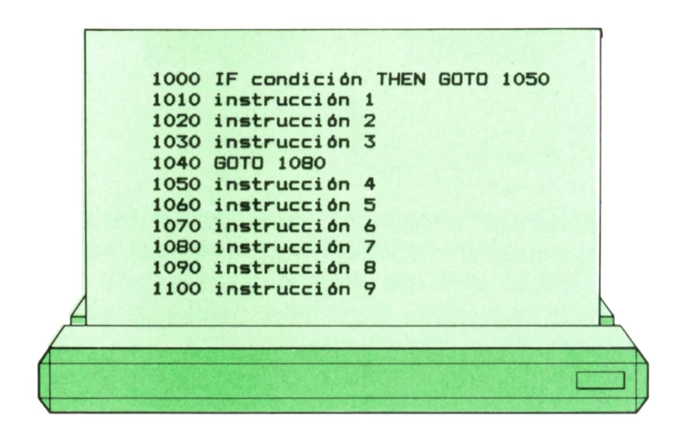
En PASCAL no será necesario hacer uso de la instrucción de transferencia incondicional en este caso. En efecto, la forma más elegante de programarlo sería la siguiente:

```
BEGIN
  IF condición THEN BEGIN
                      instrucción 4;
                      instrucción 5;
                      instrucción 6
                         END
                    ELSE BEGIN
                      instrucción 1;
                      instrucción 2;
                      instrucción 3
                         END;
  instrucción 7;
  instrucción 8;
  instrucción 9
END
```

Sin embargo, en BASIC no disponemos de la posibilidad de definir largos bloques secuenciales de instrucciones (limitados, como en PASCAL, por las palabras reservadas BEGIN-END). La única forma de definir un bloque secuencial de instrucciones consiste en apilarlas todas bajo el mismo número de instrucción, separadas por dos puntos. Si utilizáramos este modo, el organigrama anterior se convertiría en:

```
1000 IF condición THEN instrucción 4 :
     instrucción 5 : instrucción 6
     ELSE instrucción 1 : instrucción 2 :
     instrucción 3
1010 instrucción 7
1020 instrucción 8
1030 instrucción 9
```

Donde la instrucción 1000 es extremadamente larga, pues contiene más de la mitad del organigrama completo. Se trata, por tanto, de una instrucción difícil de comprender, que hace el programa poco legible y dificulta su corrección en caso de detección de errores o de que nos veamos obligados a efectuar cambios. Además, podría ser que ni siquiera tengamos la posibilidad de utilizarla, pues el número de instrucciones que debemos incluir en ella (o la longitud de cada una) podría ser tan grande que rebasemos la longitud máxima que permite utilizar nuestro intérprete BASIC para una sola instrucción. En tal caso, es mucho mejor utilizar la instrucción de transferencia incondicional de la siguiente manera:



Es evidente que este programa es mucho más fácil de comprender que el anterior, aunque ciertamente menos que el de PASCAL, que representa exactamente el organigrama.

Puede observarse, en nuestro último ejemplo, que una instrucción de transferencia incondicional puede aparecer en cualquier parte de un programa. Por ejemplo, como una de las instrucciones secundarias de una instrucción de ejecución condicional (tanto en la parte THEN como en la parte ELSE, si está presente). De hecho, la forma que hemos utilizado aquí:

IF condición THEN GOTO etiqueta

Es tan frecuente, que recibe un nombre propio «instrucción de transferencia condicional». Además, algunos intérpretes BASIC la abrevian de una de las dos formas siguientes:

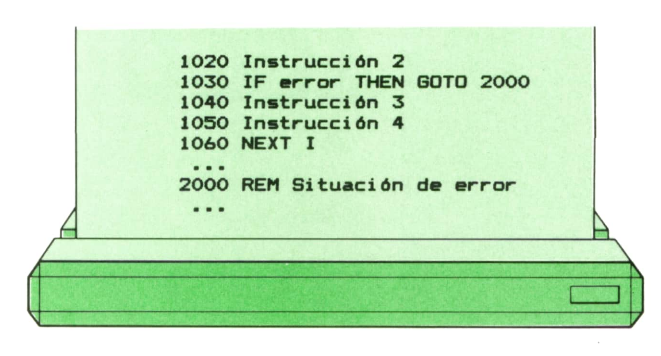
IF condición THEN etiqueta
IF condición GOTO etiqueta

Existe otro caso en que puede ser muy conveniente utilizar la instrucción de transferencia incondicional: cuando deseamos abandonar la ejecución de un bucle antes de que la condición normal de fin de bucle haya llegado a cumplirse. Esto puede suceder porque nuestro

programa haya detectado una condición de error que le exige abandonar la marcha normal del programa, para dar un mensaje de aviso e intentar recuperar una situación desde donde pueda seguir ejecutándose normalmente. Este caso es aplicable tanto a BASIC como a PASCAL.

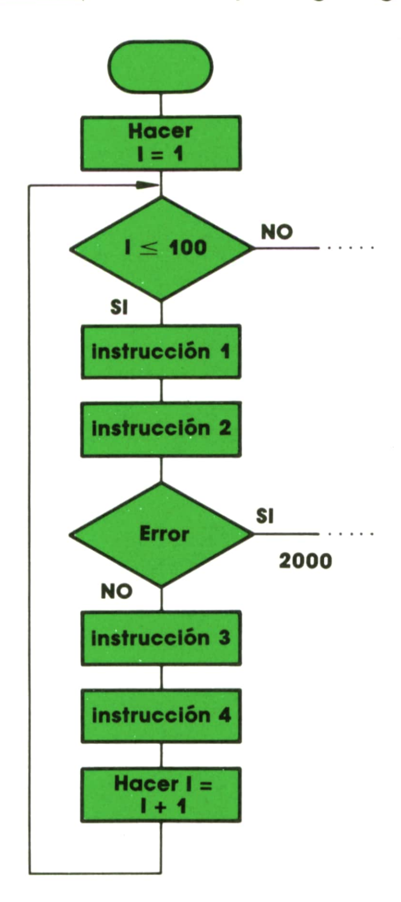
Veamos un ejemplo:

```
1000 FOR I=1 TO 100
1010 Instrucción 1
```

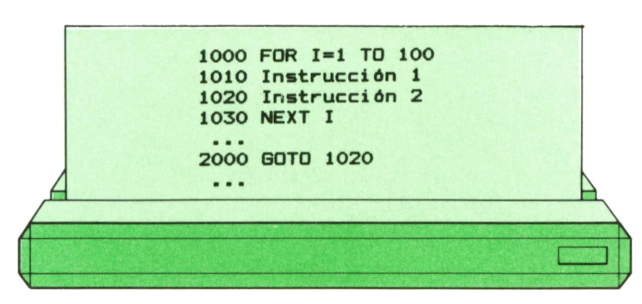


```
1020 Instrucción 2
1030 IF error THEN GOTO 2000
1040 Instrucción 3
1050 Instrucción 4
1060 NEXT I
 ...
2000 REM Situación de error
 ...
```

cuyo organigrama es:



Debe tenerse en cuenta que, aunque puede ser conveniente abandonar la ejecución de un bucle mediante una instrucción de transferencia, jamás debe penetrarse en el interior de un bucle de esa misma manera, pues podrían ocurrir efectos impredecibles. Por ejemplo:



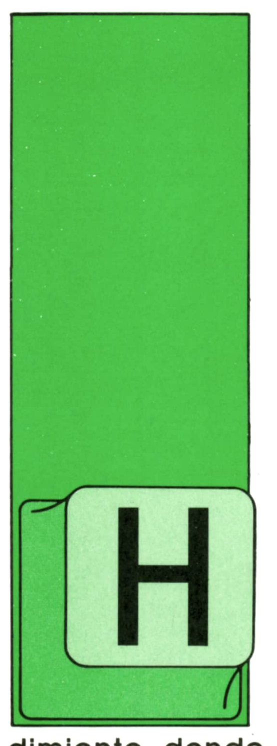
```
1000 FOR I=1 TO 100
1010 Instrucción 1
1020 Instrucción 2
1030 NEXT I
 ...
2000 GOTO 1020
 ...
```

Al ejecutar la instrucción 2000, pasaremos a la instrucción de etiqueta 1020. Esta se ejecutará correctamente, pero al llegar a la 1030, ¿qué valor tiene la variable I, que controla el bucle? No lo sabemos. Un programador hábil podría controlar incluso esto, pero en general es una práctica desaconsejable que dificulta la comprensión de los programas.

Además, algunos compiladores BASIC lo prohíben, y generan un mensaje de error si el programa llega a una instrucción NEXT sin haber pasado antes por una instrucción FOR.

LOGO

Cómo dar valor a las variables



ASTA este momento, cuando hemos necesitado utilizar una variable, hemos puesto su nombre precedido por dos puntos (:) a continuación de la definición del procedimiento donde teníamos que usarla. Después, al querer ejecutar este procedimiento escribíamos su nombre y el valor que queríamos guardar en el cajón correspondiente a la variable.

Es decir, que para conseguir que un cajón contenga un determinado valor no nos queda más remedio que utilizar esa variable junto con un procedimiento. Y esto no tiene por qué ser siempre así.

En muchos casos nos va a interesar usar variables dentro de los procedimientos sin la obligación de tener que darles valor sólo cuando ejecutemos ese procedimiento, sino que también podamos hacerlo con independencia de él. Con ello, lograremos, en la mayoría de los casos, escribir procedimientos mucho más cortos.

El comando

HAZ "nombre valor

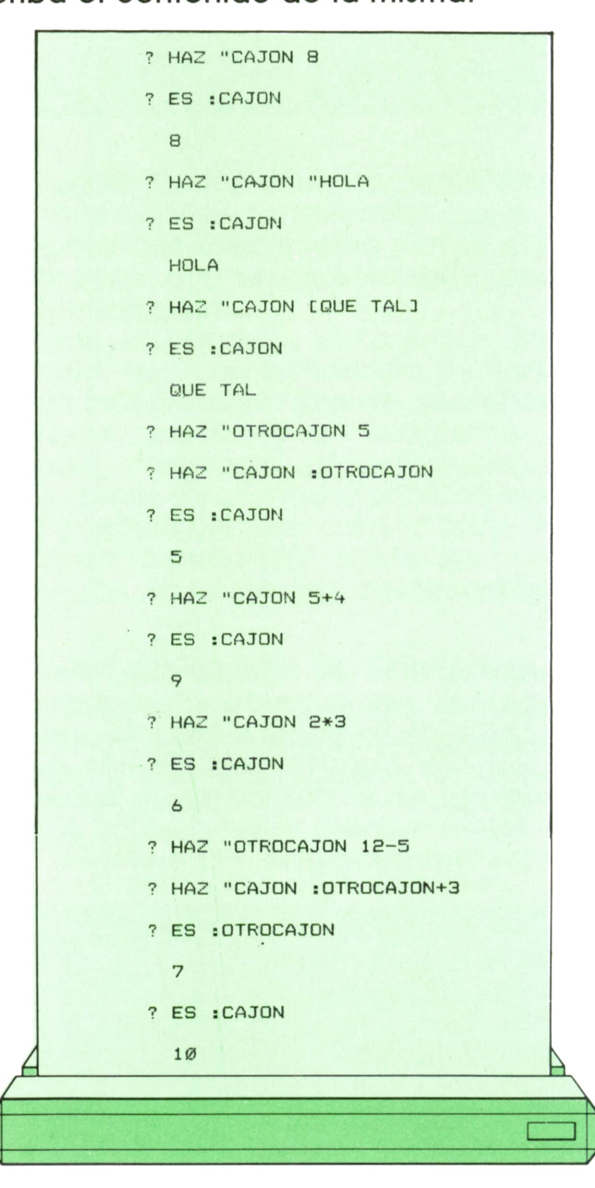
nos permite guardar en la variable llamada **nombre** lo que pongamos en **valor.**

En **valor** podemos poner todo lo que se nos ocurra:

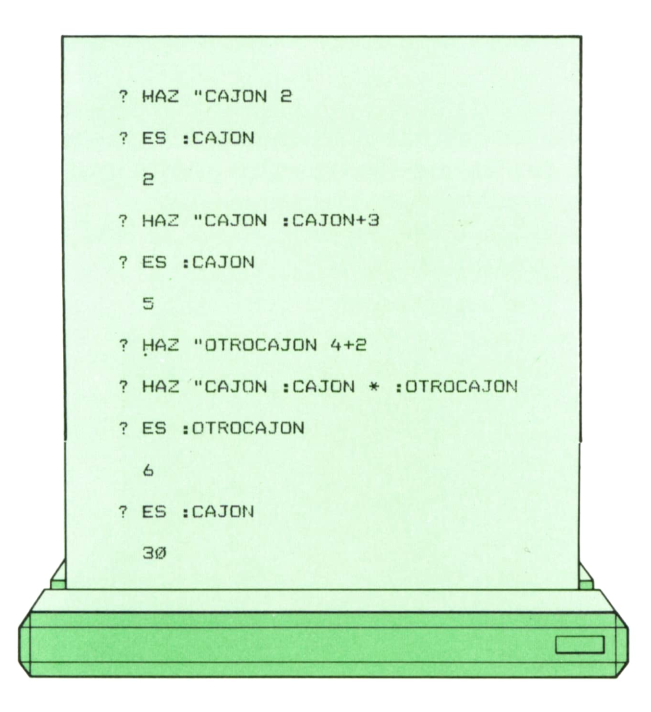
— un número
— una palabra
— una lista de palabras
— el contenido de otra variable
— el resultado de una operación

Se dice que este comando es un **comando de asignación,** ya que lo único que hace es asignar a una variable (meter en su cajón correspondiente) un determinado valor sin que esto modifique ninguna característica (posición, rumbo...) de la tortuga.

Veamos con algunos ejemplos lo que guardaría la tortuga en la variable CAJON al utilizar este nuevo comando. Para ello, tras hacer cada asignación vamos a poner el comando ES para que nos escriba el contenido de la misma:



Incluso podemos asignar como valor a una variable su propio contenido operado con otra cosa (un número, otra variable...). Por ejemplo:



Como puede verse, el utilizar el contenido de una variable para asignar valor a otra no implica que se modifique el valor de la primera, sino que éste se mantiene.



Os proponemos

1. Intenta averiguar qué valor escribiría la tortuga si después de cada comando HAZ ponemos el comando

ES :VAR

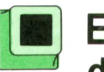
? HAZ "VAR 4
? HAZ "VAR 0
? HAZ "VAR 2 * 8
? HAZ "VAR 5 + 2 + 1
? HAZ "VAR (8 – 5) * 3
? HAZ "VAR 6 / 2

2. Ahora lo mismo con estas otras asignaciones un poco más complicadas:

a) ? HAZ "VAR1 12 – 5
? HAZ "VAR2 :VAR1
? ES :VAR1
? ES :VAR2

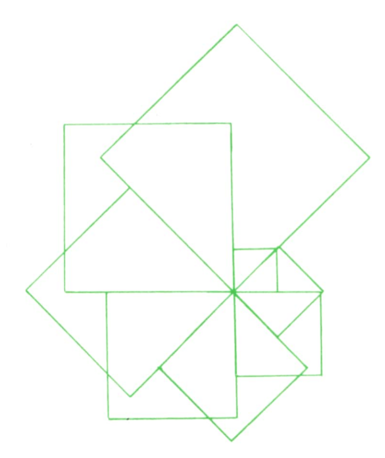
b) ?HAZ "VAR3 0
? HAZ "VAR4 :VAR3 + 5
? ES :VAR3
? ES :VAR4

c) ? HAZ "VAR5 6
? HAZ "VAR6 :VAR5 * 2
? HAZ "VAR6: VAR6/2
? ES :VAR5
? ES :VAR6
? HAZ "VAR5 0
? HAZ "VAR6 :VAR5 + 7
? ES :VAR5
? ES :VAR6
? HAZ "VAR6 :VAR6 – 3
? ES :VAR5
? ES :VAR6
? HAZ :VAR5 :VAR6
? ES :VAR5
? ES :VAR6

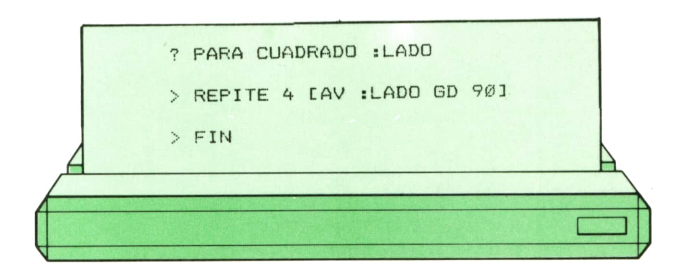


El mismo procedimiento de dos maneras

Supongamos que queremos pintar ocho cuadrados girados cada uno con respecto al anterior y que van aumentando de tamaño:

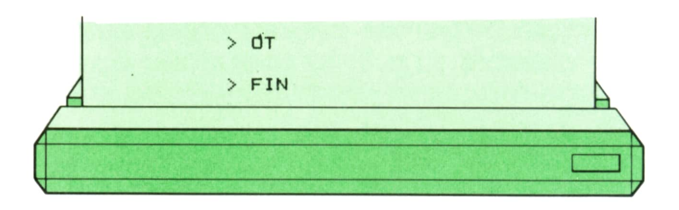


Necesitamos un procedimiento para dibujar cuadrados de tamaño variable:



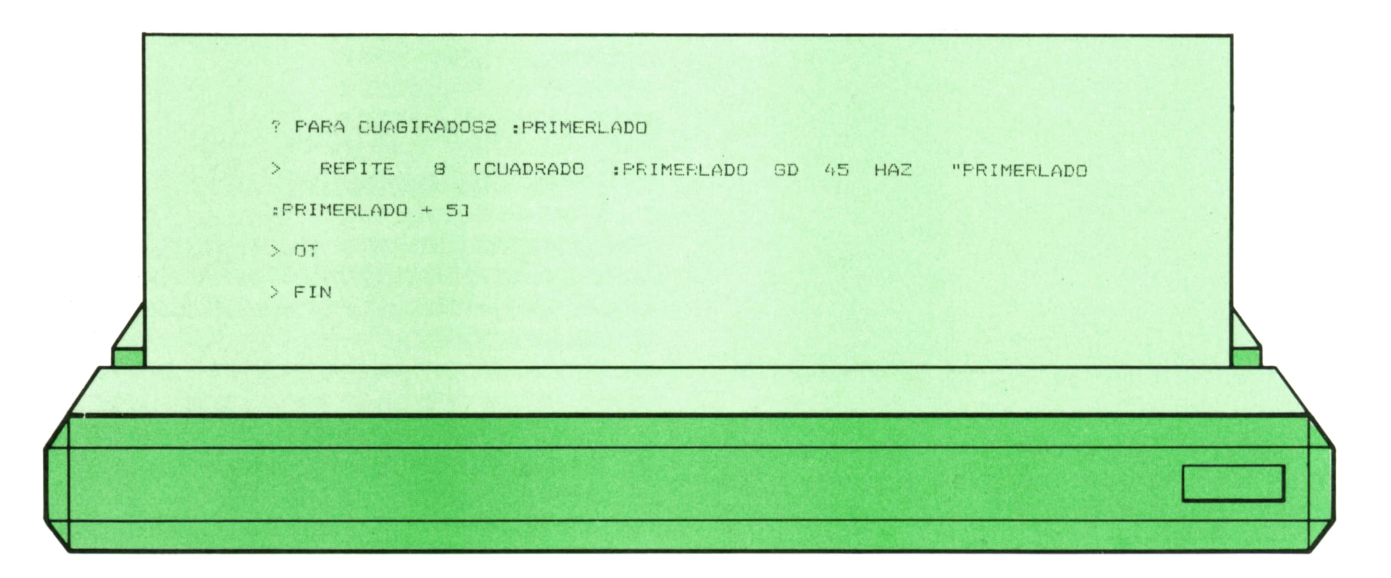
Sin utilizar el comando HAZ, el procedimiento para obtener la figura anterior sería:

```
? PARA CUAGIRADOS1
> CUADRADO 10 GD 45
> CUADRADO 15 GD 45
> CUADRADO 20 GD 45
> CUADRADO 25 GD 45
> CUADRADO 30 GD 45
> CUADRADO 35 GD 45
> CUADRADO 40 GD 45
> CUADRADO 45 GD 45
```



Como podemos ver el tamaño del lado de los cuadrados va aumentando de 5 en 5, partiendo de un valor inicial igual a 10.

Si ahora usamos el comando HAZ, se puede escribir:



Para obtener la figura tendríamos que poner:

: ? CUAGIRADOS 10

Con esto vemos que escribiendo el procedimiento de la segunda forma nos queda mucho más corto que con la primera y, además, tenemos la posibilidad de dar cualquier tamaño al primer cuadrado y, por tanto, a todos los demás.

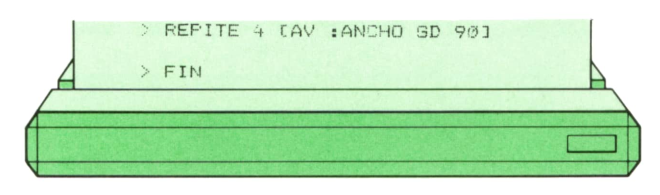


Más ejemplos con el comando HAZ

Vamos a hacer el siguiente dibujo:

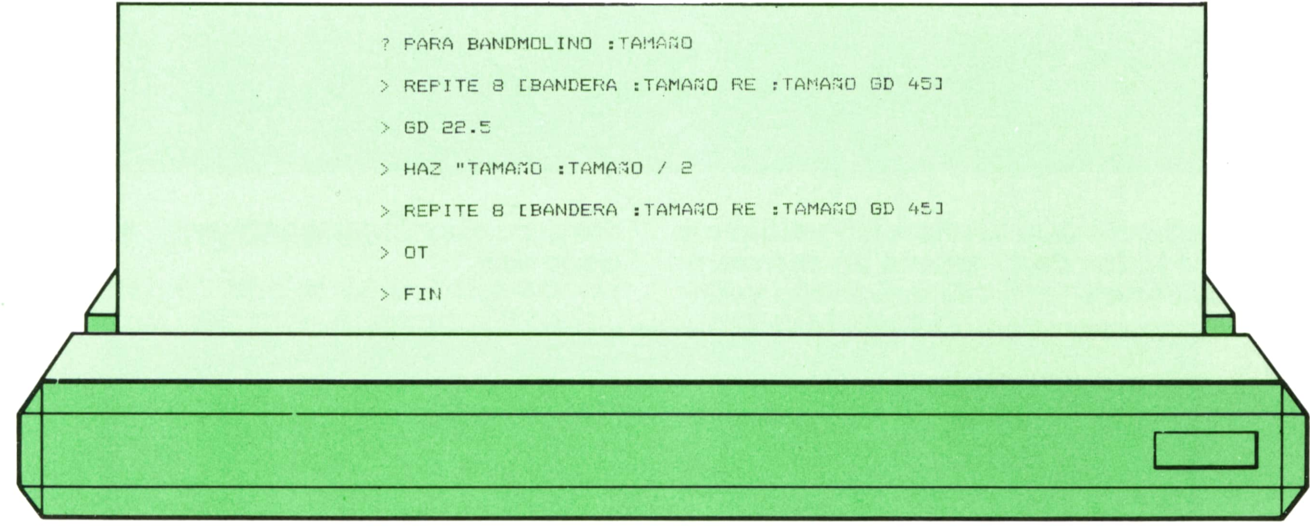
Primero tenemos que definir un procedimiento que dibuje una bandera:

```
? PARA BANDERA :ANCHO
> HAZ "LARGO :ANCHO * 2
> AV :LARGO
> RE :ANCHO
```



```
> REPITE 4 [AV :ANCHO GD 90]
> FIN
```

Ahora podemos usar este procedimiento para obtener el conjunto de las banderas:



```
? PARA BANDMOLINO :TAMAÑO
> REPITE 8 [BANDERA :TAMAÑO RE :TAMAÑO GD 45]
> GD 22.5
> HAZ "TAMAÑO :TAMAÑO / 2
> REPITE 8 [BANDERA :TAMAÑO RE :TAMAÑO GD 45]
> OT
> FIN
```

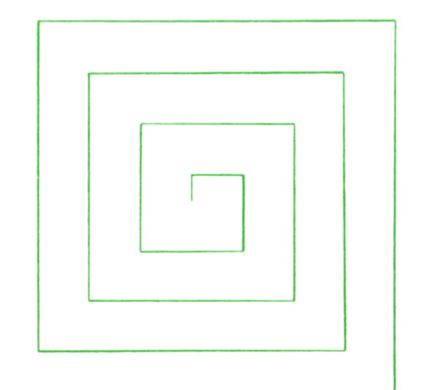
O también se puede escribir así:

```
? PARA BANDMOLINO :TAMAÑO
> REPITE 8 [BANDERA :TAMAÑO RE :TAMAÑO GD 22.5 HAZ "TAMAÑO
:TAMAÑO / 2 BANDERA :TAMAÑO RE :TAMAÑO GD 22.5 HAZ "TAMAÑO
:TAMAÑO * 2]
> OT
> FIN
```

Para ejecutarlo, podemos poner, por ejemplo:

? BANDMOLINO 60

Ahora vamos a dibujar esta espiral:



Tenemos que poner:

```
? PARA ESPIRAL :LONGITUD
> REPITE 15 [AV :LONGITUD GD 90 HAZ "LONGITUD :LONGITUD + 7]
> OT
> FIN
```

Si queremos que la diferencia entre los distintos lados de la espiral no sea siempre la misma (7), necesitamos otra variable para guardar el incremento. Nos quedaría así:

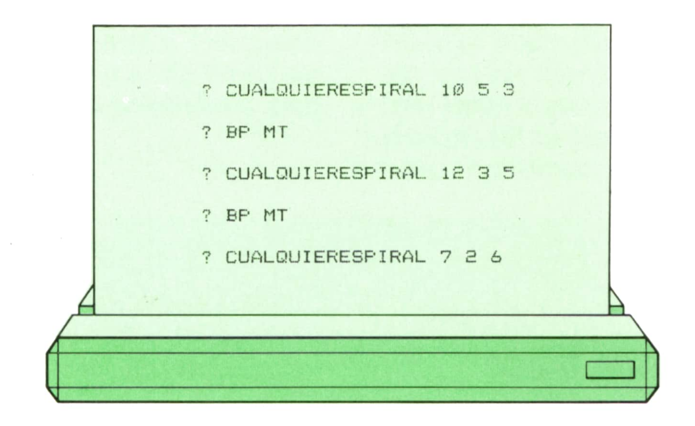
```
? PARA ESPIRAL :LONGITUD :INCREMENTO
>  REPITE  15  [AV  :LONGITUD GD 90  HAZ  "LONGITUD  :LONGITUD  +
:INCREMENTO]
> OT
> FIN
```

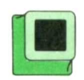
Por último, tenemos la posibilidad de hacer espirales que no sean cuadradas, sino que puedan tener varias formas (triangular, pentagonal...). Para ello usaremos otra variable que nos sirva para almacenar el número de lados correspondiente a la forma que queremos obtener:

triangular → 3
cuadrada → 4
pentagonal → 5

```
? PARA CUALQUIERESPIRAL :LONGITUD :INCREMENTO :NUMERO
>  REPITE  15  [AV  :LONGITUD GD  360  /  :NUMERO  HAZ  "LONGITUD
:LONGITUD + :INCREMENTO]
> OT
> FIN
```

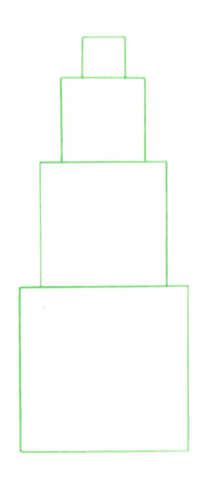
Para que la tortuga dibuje diferentes espirales, podríamos poner:



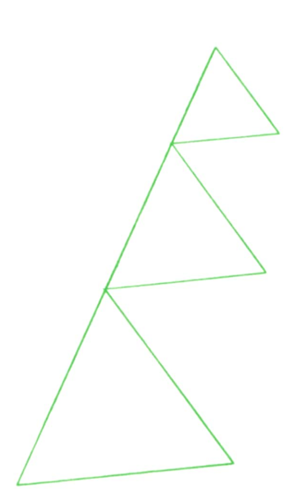


Os proponemos

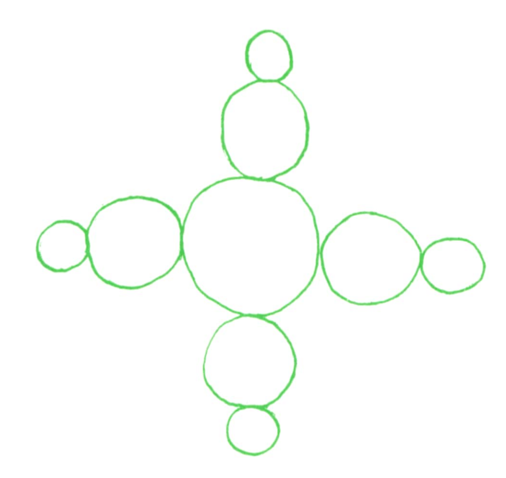
1. Intenta dibujar esta columna de cuadrados utilizando el comando HAZ:



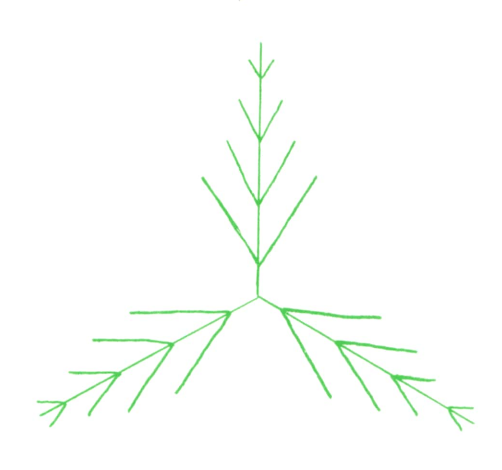
2. Ahora puedes probar con esta cadena de triángulos:



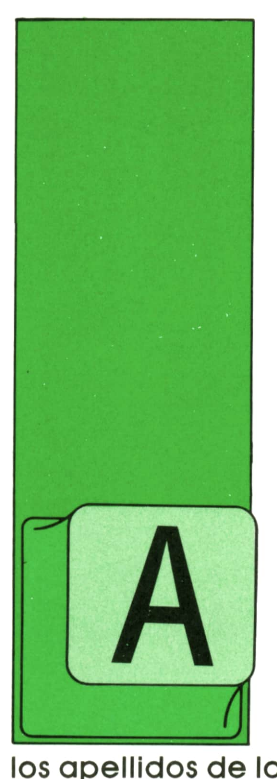
3. También puedes pintar esta figura:



4. ¿Qué te parece este trébol?



PASCAL





Ordenación de datos

A menudo se plantea el problema de tener que ordenar según un criterio dado un conjunto de datos. Por ejemplo, puede hacer falta colocar por orden alfabético los apellidos de los alumnos de un grupo, y ordenarlos en función de las notas de un examen.

Existen multitud de métodos para hacer esto y, aunque quizá veamos otros más adelante, ahora vamos a emplear el conocido como de «selección directa» que, siendo muy sencillo de entender y programar, es razonablemente rápido cuando el número de elementos a ordenar no es excesivamente alto. Consiste en lo siguiente:

Supongamos que tenemos 10 notas guardadas en la TABLA, una variable de tipo «array of real».

Se empieza por buscar la nota mayor de todas y una vez encontrada se intercambia con la que ocupaba la primera posición de TABLA. Tras este proceso TABLA contiene, evidentemente, las mismas notas que al principio, sólo que con la mayor de todas en primera posición.

A continuación se repite el mismo proceso, pero con las 9 notas restantes. Es decir, se busca la mayor de esas 9 (que están en las posiciones 2 a 10) y una vez se intercambia con la que estaba en la posición 2. Tras esto, TABLA sigue conteniendo la nota mayor en la posición 1 y además la segunda mayor en la posición 2.

Este proceso se repite para las posiciones 3 a 10 (con lo que queda la tercera nota en la posición 3), 4 a 10, etc., hasta llegar a hacerlo con las posiciones 9 a 10, momento en que queda TABLA definitivamente ordenada.

Por ejemplo, si las notas fueran:

2 7 6 9 3 1 8 4 7 5

al buscar la mayor de las diez se encuentra 9 en la cuarta posición; entonces se intercambia con la de la primera posición, 2 (la barra vertical separa las ya ordenadas de las demás):

9 | 7 6 2 3 1 8 4 7 5

Ahora se repite el proceso con todas menos la primera, permutándose, por tanto, el primer 7 con el 8 y llegándose a:

9 8 | 6 2 3 1 7 4 7 5

A continuación se repite con todas menos con las dos primeras, etc., hasta tener la TABLA ordenada. Las diferentes situaciones por las que pasaría la TABLA serían:

9 8 7 | 2 3 1 6 4 7 5
9 8 7 7 | 3 1 6 4 2 5
9 8 7 7 6 | 1 3 4 2 5
9 8 7 7 6 5 | 3 4 2 1
9 8 7 7 6 5 4 | 3 2 1 (*)
9 8 7 7 6 5 4 3 | 2 1 (*)
9 8 7 7 6 5 4 3 2 | 1

Si utilizamos el índice I para indicar qué nota se está buscando (primera, segunda...) y Total para indicar el total de notas, la estructura queda así:

Para I variando su valor desde 1 hasta Total-1 hacer:
- **Buscar la mayor nota de las comprendidas entre las posiciones I y Total y permutarla con la de la posición I.**

Cuando la mayor entre I y Total es precisamente la de la posición I, no es necesario permutarla consigo misma; son las situaciones marcadas con (*) en el ejemplo. Por tanto, la estructura quedaría mejor de esta otra manera:

Para I variando su valor desde 1 hasta Total-1 hacer:

— Buscar la mayor nota de las comprendidas entre las posiciones I y Total y si no es la de la posicion I, permutarla con ella.

En realidad, lo más corriente es que se tengan que ordenar fichas en función del valor de alguno de sus campos (las notas del ejemplo serían un campo más de las fichas de los alumnos). Si añadimos las instrucciones necesarias para leer datos del teclado y presentarlos en pantalla, tenemos el siguiente programa:

```
program Ordenar;

const
  Max   = 100;   (* Máximo número de alumnos posible *)
  Long = 12;     (* Máximo número de letras por nombre *)

type
  Nombre_t = array [1..Long] of char;
  Ficha_t = record
              Nombre,
              Apellido1,
              Apellido2   : Nombre_t;
              Nota_A,
              Nota_B,
              Nota_C   : real
            end;
  Tabla_t = array [1..Max] of Ficha_t;

var
  Grupo       : Tabla_t;
  TotalGrupo: integer;
(*----------------------------------------------------------------*)
procedure LeeDatos (var Tabla: Tabla_t; var Total: integer);

(* Pide datos y los guarda en Tabla.                    *)
(* El número de datos introducidos lo devuelve en Total *)

  var
   I : integer;
   Ok: boolean;

begin
 write ('Número de alumnos (de 1 a ',Max,'): ');
 repeat
   readln (Total);
   Ok := (1 <= Total) and (Total <= Max);
   if not Ok then write ('No vale. Repita:')
 until Ok;

 writeln ('Comience a introducir datos.');
 for I := 1 to Total do with Tabla [I] do
    begin
     writeln;
     write ('Nombre:            '); readln (Nombre);
     write ('Primer apellido:   '); readln (Apellido1);
     write ('Segundo apellido: '); readln (Apellido2);
     writeln;
     write ('Nota A: '); readln (Nota_A);
     write ('Nota B: '); readln (Nota_B);
     write ('Nota C: '); readln (Nota_C)
    end
end;
(*----------------------------------------------------------------*)
procedure Presenta (var Tabla: Tabla_t; Total: integer);

(* Presenta el contenido de TABLA.              *)
(* Se usa VAR por cuestión de tiempo y espacio. *)

  var I: integer;
begin
 for I := 1 to Total do with Tabla [I] do
   begin
    writeln;
    writeln (Nombre,' ',Apellido1,' ',Apellido2);
    writeln ('Nota A: ',Nota_A :4:1);
    writeln ('Nota B: ',Nota_B :4:1);
    writeln ('Nota C: ',Nota_C :4:1)
   end
end;

(*----------------------------------------------------------------*)
```

```
procedure Ordena (var Tabla: Tabla_t; Total: integer);

(*  Ordena el contenido de Tabla según el valor    *)
(*  de Nota_A por el método de selección directa   *)

  var
   I,J,
   IndiceMayor: integer;
   Mayor      : real;
   FichaAuxiliar: Ficha_t;

begin
 for I := 1 to Total - 1 do
   begin
   (*----------------------------------------------*)
   (* Buscar la mayor de entre I y Total.          *)
   (* En principio se toma como mayor la de índice I *)
   (* y luego se exploran las siguientes:          *)
   (*----------------------------------------------*)
    Mayor := Tabla[I].Nota_A;
    IndiceMayor := I;

    for J := I + 1 to Total do with Tabla [J] do
      if Nota_A > Mayor then   (*  <----  COMPARACION *)
         begin
          (* La mayor por ahora pasa a ser la de índice J: *)
          Mayor := Nota_A;
          IndiceMayor := J
         end;
   (*----------------------------------------------*)
   (* Si la mayor no es la de índice I, se permuta  *)
   (* su ficha con la de I.                        *)
   (*----------------------------------------------*)
    if IndiceMayor <> I then
      begin
       FichaAuxiliar := Tabla [IndiceMayor];
       Tabla [IndiceMayor] := Tabla [I];
       Tabla [I] := FichaAuxiliar
      end
   end
end;

(*------------------------------------------------------------------*)
begin
 LeeDatos (Grupo,TotalGrupo);
 Ordena   (Grupo,TotalGrupo);
 writeln ('----------------------------------');
 Presenta (Grupo,TotalGrupo)
end.
```

Si en lugar del test "Nota-A > Mayor" se hubiera utilizado "Nota-A < Mayor", las notas acabarían ordenadas de menor a mayor, pero entonces sería conveniente cambiar los nombres de Mayor e IndiceMayor por Menor e IndiceMenor, respectivamente, para que resultasen coherentes.

Si, por ejemplo, hubiese que ordenar las fichas alfabéticamente según el primer apellido, el procedimiento sería similar. La única diferencia consistiría en que habría que explorar las fichas buscando cada vez, no la de mayor Nota-A, sino la que tuviese el apellido más adelantado alfabéticamente. Entonces, el procedimiento Ordena sería:

```
procedure Ordena (var Tabla: Tabla_t; Total: integer);

(*  Ordena la variable Tabla según el contenido de  *)
(*  Apellido1 por el método de selección directa.   *)

  var
   I,J,
   IndicePrimero: integer;
   Primero      : Nombre_t;
   FichaAuxiliar: Ficha_t;
```

```
begin
 for I := 1 to Total - 1 do
   begin
   (*-----------------------------------------------------*)
   (* Buscar el primer apellido de entre I y Total.      *)
   (* En principio se toma como primero el de índice I *)
   (* y luego se exploran los siguientes:               *)
   (*-----------------------------------------------------*)
    Primero := Tabla[I].Apellido1;
    IndicePrimero := I;

    for J := I + 1 to Total do with Tabla [J] do
      if Antes (Apellido1, (* que *) Primero) then
        begin
         (* El primero por ahora pasa a ser el de índice J: *)
         Primero := Apellido1;
         IndicePrimero := J
        end;
   (*-----------------------------------------*)
   (* Si el primero no es el de índice I,   *)
   (* se permuta su ficha con la de I.      *)
   (*-----------------------------------------*)
    if IndicePrimero <> I then
      begin
       FichaAuxiliar := Tabla [IndicePrimero];
       Tabla [IndicePrimero] := Tabla [I];
       Tabla [I] := FichaAuxiliar
      end
   end
.end;
```

Como se ve, el procedimiento es prácticamente idéntico al de las notas, sólo que adaptado al nuevo tipo de campo y con algunos nombres distintos para que resulte más claro.

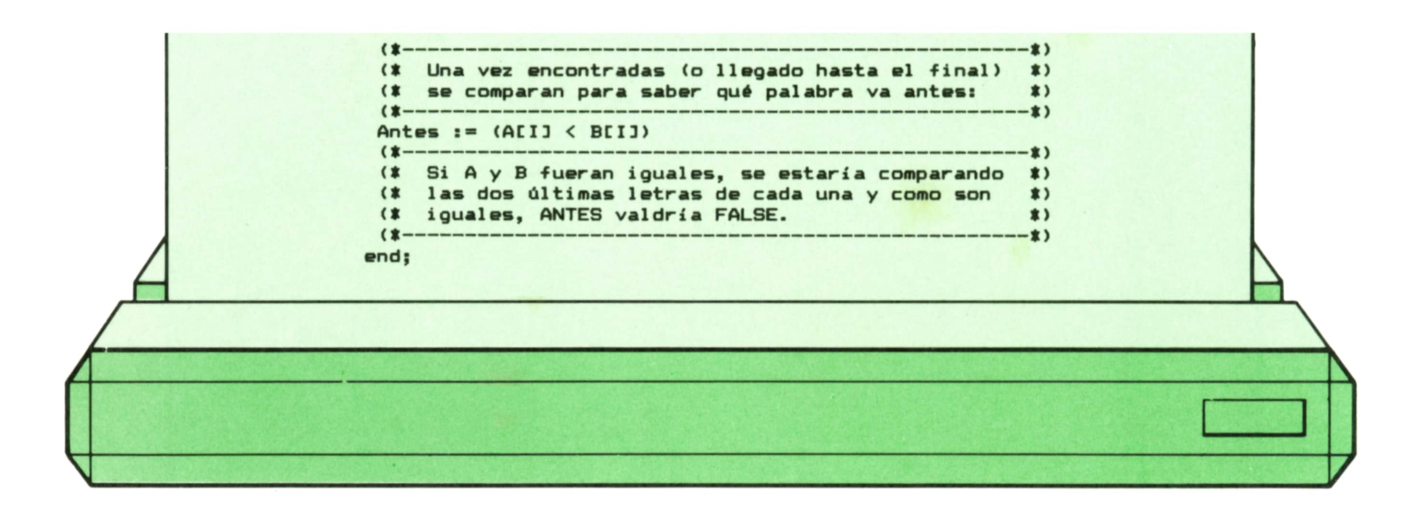
Dados dos números N1 y N2, para saber cuál está por delante basta con evaluar la expresión lógica "N1 > N2". Sin embargo, dadas dos palabras, para saber cuál es la que va por delante alfabéticamente el proceso es un poco más complejo: hay que compararlas letra a letra hasta encontrar el primer par de letras distintas entre sí y una vez localizadas mirar a ver cuál va antes alfabéticamente. En caso de llegarse al final de alguna de las palabras sin haber encontrado ninguna diferencia, se considera que la más corta va por delante; no obstante, dado que las palabras se guardan en las variables complementándolas por la derecha con espacios en blanco o chr (0), y que estos caracteres, en general, están alfabéticamente por delante de cualquier letra, es posible explorar los dos "array of char" de principio a fin sin preocuparse de las longitudes reales de las palabras.

En el último procedimiento Ordena se utiliza la función ANTES para hacer estas comparaciones; deberá ubicarse en el programa por delante de Ordena o, mejor aún, dentro de él:

```
function Antes (A,B: Nombre_t): boolean;

(* Devuelve TRUE si A va por delante de B alfabéticamente *)

 var I: integer;
begin
 (*----------------------------------------------*)
 (*  Se exploran las dos palabras de izquierda a  *)
 (*  derecha buscando la primera letra distinta:  *)
 (*----------------------------------------------*)
 I := 0;
 repeat I := I + 1 until (A[I] <> B[I]) or (I = Long);
```



```
(*------------------------------------------------*)
(*  Una vez encontradas (o llegado hasta el final)  *)
(*  se comparan para saber qué palabra va antes:    *)
(*------------------------------------------------*)
Antes := (A[I] < B[I])
(*------------------------------------------------*)
(*  Si A y B fueran iguales, se estaría comparando  *)
(*  las dos últimas letras de cada una y como son   *)
(*  iguales, ANTES valdría FALSE.                   *)
(*------------------------------------------------*)
end;
```

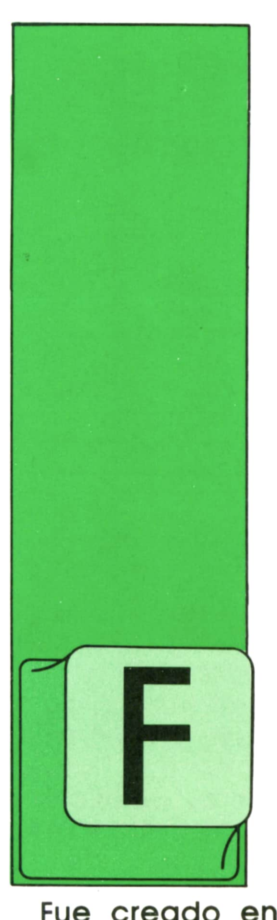
Probablemente sería bueno pasar las letras a mayúsculas antes de compararlas para evitar sorpresas; para ello se podría utilizar la función MAYUSCULA que ya conocemos.

(NOTA: Con el tipo no estándar «string» disponible con algunos compiladores sí es posible comparar directamente dos palabras.)

En definitiva, el procedimiento ORDENA se puede adaptar a prácticamente cualquier tipo de variable, con tal de disponer de alguna función que nos permita decidir de entre dos valores cuál va por delante. Por otra parte, si en el programa del ejemplo hubiese que añadir más campos a las fichas, lo único que habría que cambiar, además de su definición, serían los procedimientos Presenta y LeeDatos.

OTROS LENGUAJES

FORTRAN



ORTRAN es el nombre de un lenguaje de alto nivel orientado a cálculos científicos. De ahí su nombre: FORmula TRANslator, o traductor de fórmulas.

Fue creado en la compañía IBM en 1956 y ha ido evolucionando a lo largo de los años, dando lugar a diferentes versiones, entre las que destacan FORTRAN II y FORTRAN IV.

Todas estas versiones tienen en común una serie de instrucciones delimitadas y definidas por la ANS (American National Standard), que trata de homologar los lenguajes de programación.

Los programas que se muestran como ejemplo en estos fascículos han sido elaborados en un IBM PC con FORTRAN IV de MICROSOFT.

Los temas que se desarrollarán son, a grandes rasgos:

— Definición de variables.
— Sentencias de entrada/salida.
— Operaciones aritméticas.
— Instrucciones condicionales:
— Bucles.

que permiten al lector obtener una primera visión del FORTRAN



Normas de escritura

Las 80 columnas de la pantalla se subdividen de la forma siguiente:

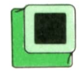
Las cinco primeras se utilizan para escribir los números de sentencia, que permiten identificar una determinada instrucción. En otros lenguajes se denominan etiquetas (label).

Si en la columna 6 aparece un carácter cualquiera, indica la continuación de la línea anterior.

Las instrucciones se deben escribir entre las columnas 7 y 72.

Si en vez de una sentencia ejecutable se desea escribir una línea de comentario, ignorada por el compilador, se debe escribir una «C» en la columna 1.

Las restantes columnas (73-80) que se empleaban anteriormente para numerar las sentencias han perdido ahora su significación.



Nombres FORTRAN

Un nombre válido para el Fortran es una palabra con una longitud máxima de 6 caracteres. El primero debe ser una letra, mientras que los restantes pueden ser un dígito o letra.

El primer carácter sirve para identificar el tipo de la variable.

Si empieza por I, J, K, L, M o N, contendá valores enteros.

Si el primer carácter está en el rango A-H o 0-Z, se trata de una variable real.

Según estas normas, las variables AREA y RADIO serán reales y podrán contener

valores reales, mientras que MINIMO sólo podrá almacenar números enteros.

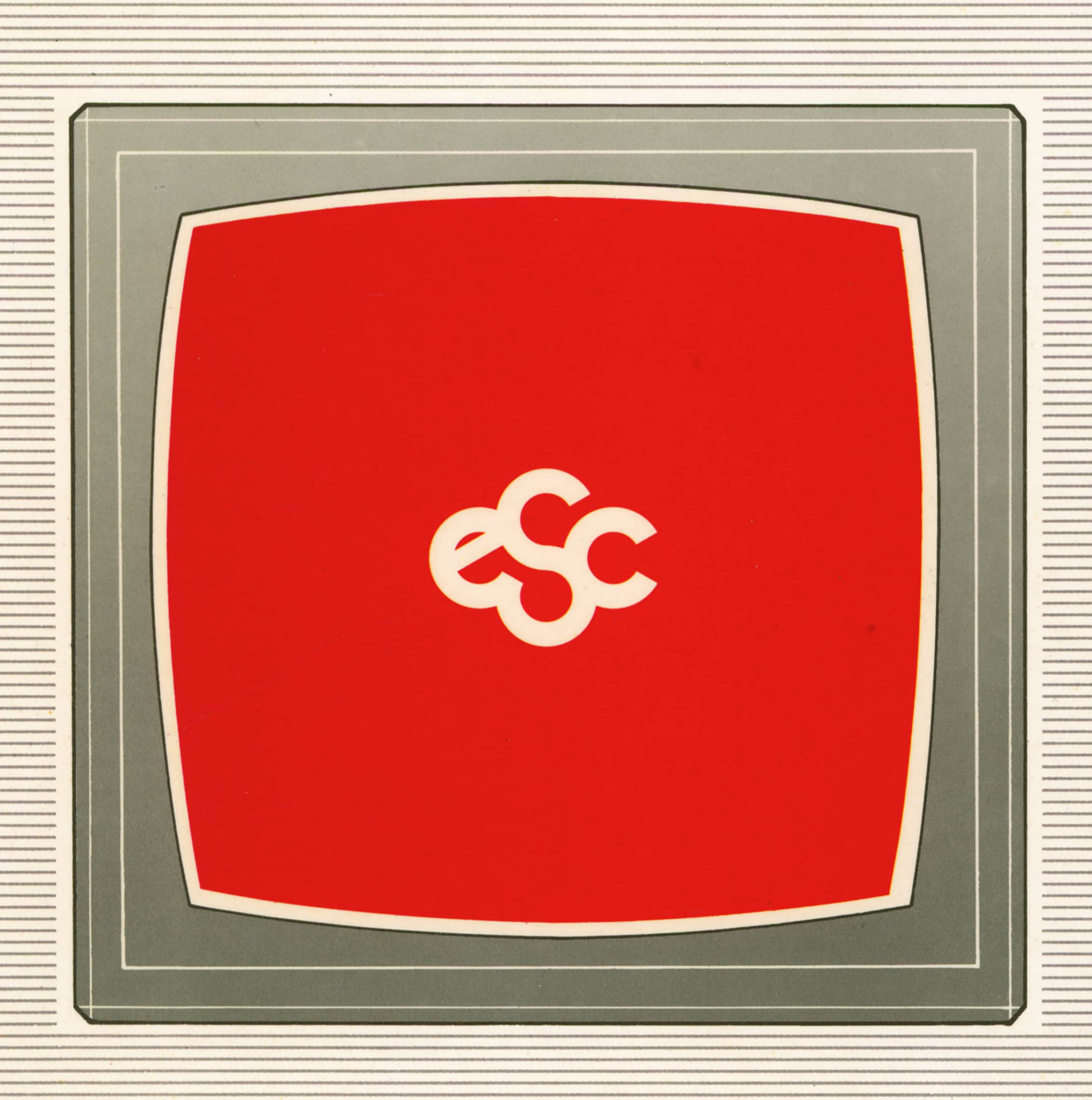
Existe una sentencia que permite definir una variable, independientemente de su inicial:

```
INTEGER
REAL      variable-1, variable-2,...
LOGICAL
```

Esta sentencia debe ir al comienzo del programa.

```
C PROGRAMA QUE CALCULA EL AREA DE UN TRIANGULO

      INTEGER BASE, ALTURA
      WRITE ( 0, 100 )
100   FORMAT ( 1X, 'INTRODUZCA BASE Y ALTURA DEL TRIANGULO ' )
      READ ( 0, 200 ) BASE, ALTURA
200   FORMAT ( I2, 1X, I2 )
      AREA = BASE * ALTURA / 2.0
      WRITE ( 0, 300 ) AREA
300   FORMAT ( 1X, 'EL RESULTADO ES  ', F4.1 )
      STOP
      END
```



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼